ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Implementierung einer Management-Komponente zur flexiblen Internationalisierung existierender Anwendungen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Janosch Walter Zoller
janosch.zoller@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Dr. Johannes Schobel

2018

Fassung 22. Oktober 2018

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Oft kommt es vor, dass bei einem Softwareprojekt manche Anforderungen erst nach der eigentlichen Implementierungsphase aufgedeckt werden – diese müssen dann teuer nachgerüstet werden. So geht es auch manchen Anwendungen, die während ihrer Entwicklung immer weiter und weiter wachsen und auf einmal vor ganz neuen Herausforderungen stehen. Wenn eine Anwendung grenzübergreifend und kollaborativ benutzt werden soll, kommt schnell die Problematik einer Sprachbarriere ins Spiel, die so zuvor niemand vorhergesehen hat.

In der vorliegenden Arbeit soll es darum gehen, mittels einer Angular-Bibliothek eine Lösung für dieses Problem zu erarbeiten. Die Bibliothek soll die Internationalisierung für bestehende Anwendungen vollständig übernehmen und dabei den Aufwand für die Entwickler der bestehenden Anwendung beim initialen Einbau der Bibliothek sowie später in der Wartung so gering wie möglich halten. Begleitend zu dieser Arbeit wird ein lauffähiger Prototyp erstellt, der hinsichtlich seiner Funktionsweise und den getroffenen Designentscheidungen dokumentiert und analysiert wird.

Danksagung

Ich danke meinem Betreuer, Dr. Johannes Schobel, für die große Unterstützung und Flexibilität, mit der er mich bei der Themenwahl und während der Erstellung dieser Arbeit unterstützt hat, sowie für das große Verständnis, das er meiner speziellen Lebenssituation entgegenbrachte.

Dr. Alexander Raschke danke ich ganz herzlich für seine Offenheit und Beratung im Vorfeld.

Ein großer Dank geht an Felix Riesterer und SELFHTML, ohne die ich meine Begeisterung für die Webprogrammierung in dieser Form vermutlich nicht gefunden hätte – selbstverständlich auch für die vielen Gespräche und das große Interesse an meiner Arbeit.

Aber vor allem möchte ich meiner Familie danken – meinen Eltern, meinen Töchtern und meiner Partnerin, die mich durch die Zeit großer Doppelbelastung hinweg ertragen haben und für eine viel zu lange Zeit hinter Studium und Beruf zurückstecken mussten.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Problemstellung 2

1.2 Zielsetzung 3

1.3 Struktur der Arbeit 4

2 Konzeption und Vorüberlegungen **5**

2.1 Anforderungsanalyse 5

2.1.1 Funktionale Anforderungen mit User Stories 6

2.1.2 Nicht-Funktionale Anforderungen 11

2.2 Testumgebung: Casa del Diavolo 14

2.3 Die Grundstruktur 17

2.4 Umgang mit Alternativsprachen 25

3 Implementierung und Dokumentation **29**

3.1 Das Grundgerüst der Bibliothek 29

3.2 Rückschlüsse auf die REST-Schnittstelle 30

3.3 IntmanSwitcher in variabler Ausgestaltung 31

3.4 Verwaltung mit AdminComponent 33

3.5 Sprachverwaltung mit AdminLanguageComponent 37

3.6 Übersetzungen verwalten mit AdminTranslationComponent 38

3.7 Textdaten identifizieren und ersetzen 40

3.8 Die zentrale Datenverwaltung 44

4 Ausblick und Schlussbemerkungen **45**

4.1 Bislang nicht erfüllte Anforderungen 45

4.2 Schlusswort 47

A Quelltexte **51**

A.1 Basisversion – Testumgebung Casa del Diavolo 51

A.2 Die Quelltexte der Bibliothek 53
A.2.1 Modul und Public-API der Bibliothek 53
A.2.2 Klassen zur Repräsentation von Daten 55
A.2.3 Die Komponente SwitcherComponent 56
A.2.4 Die Komponente AdminComponent 59
A.2.5 Interne Komponente AdminLanguageComponent 66
A.2.6 Interne Komponente AdminTranslationComponent 70
A.2.7 Die Direktive idDirective 79
A.2.8 Die Klasse TextContainer 80
A.2.9 Die Klasse TextualContent 89
A.2.10 Zentraler Service IntManLibService 94
A.3 Die HttpClientInMemoryWebApi der Testumgebung 103
A.4 Finale Version – Testumgebung Casa del Diavolo 106

1

Einleitung

„Die Grenzen meiner Sprache
bedeuten die Grenzen meiner Welt.“

Ludwig Wittgenstein (1889-1951)

Im Zeitalter des World Wide Web und der digitalen Kommunikation, des Freihandels und der multinationalen Unternehmen, der weltweiten Vernetzung und des globalen Austauschs gehören Grenzen eigentlich nicht in unseren alltäglichen Sprachgebrauch – daran haben auch die politischen Entwicklungen in Richtung der nationalen Kleingeisterei, die sich in den letzten Jahren verstärkt abzeichnen, nichts geändert.

Der österreichische Philosoph Ludwig Wittgenstein ist in einer anderen Zeit aufgewachsen, hat zerfallende Großreiche, internationale Spannungen und zwei Weltkriege miterlebt. Grenzen waren für ihn sicher eine alltäglichere Erfahrung als für uns heute.

Doch es sind nicht die nationalen Grenzen, von denen er in seinem Zitat spricht, sondern es sind geistige Grenzen, Grenzen der Kommunikation, an die wir auch heute noch tagtäglich stoßen können.

Meistens nennen wir sie nicht Grenzen, sondern Barrieren. Sprachbarrieren.

Was nützt es, wenn das Internet voll mit Informationen ist, wenn ich die Sprache oder Schrift nicht verstehe, in der die Informationen aufbereitet sind? Die internationale Kommunikation und die internationale Kooperation bringen es mit sich, dass man heutzutage oft aus seiner gewohnten Umgebung ausbrechen muss, oder möchte, um am internationalen Austausch teil zu haben. Doch wir haben keine Weltsprache, die von allen Menschen gleichermaßen gesprochen wird – und am wohlsten fühlen sich die

meisten Menschen vermutlich trotz guter Englisch- oder Spanischkenntnisse dann, wenn sie Inhalte in der Sprache aufbereitet bekommen, die sie am besten und am liebsten sprechen.

Die Internationalisierung, im Netzjargon oft auch als *i18n* abgekürzt, ist daher schon seit längerer Zeit ein großes Thema im Internet und bei den Verantwortlichen in der Standardisierungsorganisation W3C, doch nicht nur da.

Auch überall dort, wo über nationale Grenzen hinweg gelebt, gearbeitet oder auch geholfen wird, kommt man mit Menschen in Berührung, die hinter ihren eigenen Sprachbarrieren stecken und deshalb Werkzeuge zur Selbstbestimmung, zur Verbesserung ihrer Lebensumstände oder auch zur Kollaboration mit Anderen nicht nutzen können.

Daher besteht ein ständiger Bedarf an Konzepten zur Internationalisierung und entsprechender Software, die eine Anwendung dazu befähigt, von Menschen auch über Sprachbarrieren hinweg jeweils in ihrer Sprache benutzt zu werden.

Mit dieser Arbeit soll in diesem Sinne ein kleiner Beitrag in diese Richtung geleistet werden.

1.1 Problemstellung

Im Bereich der Frameworks, die gleichermaßen für mobile Anwendungen wie auch für Webseiten eingesetzt werden können, gehört das Framework **Angular** von Google zu den populärsten Frameworks überhaupt.

So wird Angular in vielen Bereichen eingesetzt – beispielsweise auch in Apps und Software, die daraufhin grenzübergreifend vertrieben und benutzt werden sollen.

Beispielsweise wird an der Universität Ulm momentan im Rahmen des Projekts *QuestionSys* an einem System geforscht, mit dem psychologische Umfragen verwaltet werden können. Dabei ist es bereits möglich, im Rahmen der Erstellung eines Fragebogens entsprechend verschiedene Sprachen bereitzustellen, die dann den Teilnehmern an der Befragung zur Verfügung stehen.

Nun kann es aber nicht nur auf Seiten der Teilnehmer einer solchen Befragung dazu kommen, dass Menschen mit unterschiedlichen Sprachen den selben Fragebogen ausfüllen sollen, sondern es kann natürlich auch im Rahmen einer internationalen Zusammenarbeit dazu kommen, dass die Fragebogen von Teams betreut werden, die eben nicht alle dieselbe Sprache sprechen.

Während es bereits sehr lange hinreichend gute Konzepte zur Internationalisierung und multilingualen Bereitstellung von Frontends gibt und diese Anforderung meist auch von Beginn an in Softwareprojekten berücksichtigt wird, erfahren viele Backends hier eine eher stiefmütterliche Behandlung.

Auch Angular besitzt zwar eigene Funktionalitäten zur Internationalisierung, ermöglicht aber durch die konkrete Implementierung nur, dass ganze Anwendungen effizient übersetzt und dann nach Sprache getrennt einzeln zur Verfügung gestellt werden. Das ist keine Lösung des Problems, das sich ergibt, wenn Menschen mit ganz unterschiedlichen Sprachkenntnissen eine einzelne Plattform bedienen sollen.

Leider kann man bei existierenden Systemen auch nicht davon ausgehen, dass es einfach so möglich wäre, die komplette Anwendung entsprechend zu überarbeiten – zu groß sind die notwendigen Eingriffe, die am System vorgenommen werden müssen und zu gering der wirtschaftliche oder ideologische Ertrag.

Um diese Probleme zu lösen benötigt es eine Bibliothek, die in existierende Anwendungen eingebunden werden kann und dort die Internationalisierung übernimmt. Gleichzeitig aber soll dabei für die Administratoren und Entwickler der bestehenden Anwendung der Aufwand zum initialen Einbau der Bibliothek und damit letztlich auch zum Einbau der Funktionalität zur Internationalisierung so gering wie nur möglich gehalten werden.

1.2 Zielsetzung

Gemäß der oben beschriebenen Problemstellung soll in der vorliegenden Arbeit eine solche Bibliothek konzipiert und in Teilen implementiert werden.

Es geht dabei darum, im Sinne eines *proof of concept* grundlegende Basisfunktionalitäten bereitzustellen und die grundlegende Machbarkeit zu untersuchen.

Die vorliegende Arbeit dokumentiert dabei sowohl Teile der Konzeption als auch Teile der Implementierung, um den grundlegenden Gedanken hinter der konkreten Implementierung herauszuarbeiten und auf Details der Implementierung hinzuweisen.

1.3 Struktur der Arbeit

Auf diese Einleitung folgt Kapitel 2, *Konzeption und Vorüberlegungen*. Dort sollen die wichtigsten Gedanken gesammelt werden, die noch vor der Implementierung von Bedeutung sind. Insbesondere wird dort auch noch einmal eine Anforderungsanalyse dargestellt, die die oben dargestellte Zielsetzung und Problemstellung um wichtige Details ergänzt und konkretisiert.

Kapitel 3, *Implementierung und Dokumentation*, analysiert die konkrete Implementierung, die im Rahmen dieser Arbeit entstanden ist, und verweist an den jeweils geeigneten Stellen auf die im Anhang abgedruckten Quelltexte. Hier wird die Verteilung der verschiedenen Funktionalitäten auf die einzelnen Teile der Bibliothek vorgestellt und diskutiert.

Schließlich wird in Kapitel 4, *Ausblick und Schlussbemerkungen*, noch einmal Resümee gezogen sowie die Erfüllung von Zielsetzung und Anforderungen diskutiert.

2

Konzeption und Vorüberlegungen

Im vorliegenden Kapitel soll die Konzeption und die Vorüberlegungen zur Implementierung der Software in Auszügen vorgestellt werden. Dabei soll zunächst in Abschnitt 2.1 in Form einer Anforderungsanalyse aufgedeckt und formalisiert werden, was die zu programmierende Software leisten können soll. Daraufhin werden die aufgrund dieser konkreten Anforderungen notwendigen Maßnahmen analysiert. So stellt Abschnitt 2.2 dar, in welchem Kontext die Implementierung der Software gemäß den Anforderungen zu erfolgen hat. Abschnitt 2.3 behandelt verschiedene Möglichkeiten zur Umsetzung der wichtigsten Anforderungen in Bausteinen der im Entstehen begriffenen Software und nimmt unter diesen eine Abwägung auf Basis der Anforderungen vor. Abschnitt 2.4 schließlich diskutiert eine organisatorische Frage zur Datenhaltung auf der selben Basis. Auf eine umfassende und ausführliche Darstellung aller Vorüberlegungen in Form eines Lastenhefts soll an dieser Stelle ausdrücklich verzichtet werden. Die Inhalte dieses Kapitels stellen dann die Grundlage für die konkrete Implementierung dar, über die im darauf folgenden Kapitel berichtet wird.

2.1 Anforderungsanalyse

Wie [1, Kapitel 1.1] bezüglich der Anforderungsanalyse schreibt, ist es ein fundamentaler Teil der Software-Konzeption, die Anforderungen (engl. *requirements*) an ein Software-Produkt zu entdecken und zu identifizieren. Nur der Vergleich der fertigen Software mit den im Vorhinein gestellten Anforderungen an die Software machen letztendlich entscheidbar, ob die Software korrekt implementiert wurde oder nicht.

Neben der Messbarkeit der korrekten Implementierung, die über die Anforderungsanalyse hergestellt werden soll, erfüllt die Anforderungsanalyse noch eine weitere sehr wichtige Funktion: Eine ordentlich dokumentierte Anforderungsanalyse stellt als Teil eines Lastenhefts sicher, dass keine Missverständnisse über das zu entwickelnde System entstehen – weder zwischen Auftraggeber und Entwicklern, noch innerhalb des Entwicklerteams. Am Ende der Anforderungsanalyse sollen die Anforderungen vollständig sein, d.h. es gibt keine nicht genannten Erwartungen des Auftraggebers an das zu entwickelnde System mehr, sie sollen konsistent sein (oft sind Wunschvorstellungen eines Auftraggebers noch mit Widersprüchen behaftet) und sie sollen präzise definiert und dadurch letztlich auch nachprüfbar sein.

Die Vermeidung von Missverständnissen spielt im vorliegenden Fall zwar eine Rolle, aber keine so bedeutsame wie in einem echten Kunde-Dienstleister-Verhältnis. Einerseits ist im vorliegenden Fall kein Entwicklungsteam, sondern eine Einzelperson mit der Entwicklung betraut, andererseits haben Problemstellung und Auftraggeber bereits einen deutlichen technischen Hintergrund, was die Gefahr für Missverständnisse oder Widersprüche verringert. Diese Überlegung führt dazu, dass im vorliegenden Fall kein detailiertes Lastenheft vorgelegt wird. Sie schmälert aber nicht die Bedeutung der Anforderungen für das Softwareprojekt als ganzes, und damit auch für die Dokumentation an dieser Stelle.

Insbesondere weist auch [2, Kapitel 4.1] darauf hin, dass Schwächen in der Anforderungsanalyse Entwicklungsaufwand von der Konzeptions- und Implementierungsphase in die Instandhaltungsphase verschieben. Dies führt in vielen Fällen zu einem großen Schaden, einerseits hinsichtlich Aspekten des Aufwands, andererseits aber auch hinsichtlich der Reputation der Software als Ganzes. Deshalb soll im Folgenden die Anforderungsanalyse besonders sorgfältig in den Fokus gerückt werden.

2.1.1 Funktionale Anforderungen mit User Stories

Funktionale Anforderungen treffen Aussagen über die Funktionalität des Systems, sie benennen also einzelne Fähigkeiten, über die das fertige System verfügen soll. Sie sind dabei soweit möglich unabhängig von der verwendeten Technologie und lassen Qua-

litätsaspekte zunächst außer Acht. Für kleinere Projekte mit verhältnismäßig geringer Dokumentation (dort als *rabbit projects* bezeichnet), empfiehlt [1, Kapitel 10], einen Zugang zu den funktionalen Anforderungen über *user stories* zu wählen. Die Funktionalen Anforderungen sollen daher nun anhand der verschiedenen Benutzungsarten, die das System ermöglichen soll, gruppiert werden.

Als Benutzer der existierenden Anwendung...

- möchte ich alle Texte im Userinterface der existierenden Anwendung in meiner Sprache präsentiert bekommen, damit ich die Funktionsweise der Anwendung genau verstehe.
- möchte ich die Interface-Texte, die es in meiner Sprache nicht gibt, in einer anderen Sprache angezeigt bekommen, die ich so gut wie möglich verstehe.
- möchte ich die Möglichkeit haben, Interface-Texte, die ich nicht verstehe, in einer anderen Sprache anzeigen zu lassen.
- möchte ich, dass Datum-, Uhrzeit- und andere Angaben so formatiert werden, dass sie zur Sprache des umliegenden Texts passen.

Aus dieser *user story* aus Perspektive eines Benutzers ergeben sich die folgenden funktionalen Anforderungen:

FA101 – Ausliefern internationalisierter Textbausteine: Das System liefert Interface-Texte, z.B. Beschriftungen von Buttons, Überschriften und erläuternde Inhalte, in einer vorausgewählten Sprache aus, so dass die existierende Anwendung diese anzeigen kann.

FA102 – Ausliefern von Textbauteilen in alternativer Sprache: Das System liefert Interface-Texte in einer alternativen Sprache aus, sofern sie in der vorausgewählten Sprache nicht existieren.

FA103 – Angemessene Wahl einer alternativen Sprache: Das System hat Kenntnis darüber welche alternativen Sprachen höchstwahrscheinlich durch den Anwender verstanden werden und wählt eine alternative Sprache auf Basis dieser Kenntnis aus.

FA104 – Manuelle Sprachwahl: Das System ermöglicht es dem Benutzer, die vorausgewählte oder automatisch ausgewählte alternative Sprache durch eine manuelle Einstellung zu überschreiben und liefert den Interface-Text dann in der manuell ausgewählten Sprache aus.

FA105 – Lokalisierung passend zur Sprache eines Texts: Das System ermöglicht die korrekte Lokalisierung von automatisch formatierten Angaben im Interfacetext passend zur Sprache des ausgelieferten Interface-Textbausteins.

Als Administrator der existierenden Anwendung...

- möchte ich die Übersetzungen der Interface-Textbausteine für die einzelnen Sprachen bearbeiten können.
- möchte ich Interface-Textbausteine verwalten können, um Altlasten zu entfernen und eventuell fehlende Textbausteine nachzutragen.
- möchte ich auf fehlende Übersetzungen für Interface-Textbausteine aufmerksam gemacht werden, damit mir keine fehlenden Übersetzungen entgehen.
- möchte ich bei fehlenden Übersetzungen zwischen einer manuell eingetragenen Übersetzung, der Anzeige in einer alternativen Sprache und durch das System unterbreiteten Übersetzungsvorschlägen wählen können, so dass einzelne fehlende Übersetzungen mit möglichst geringem Aufwand kompensiert werden können.
- möchte ich verfügbare Sprachen hinzufügen oder entfernen können.
- möchte ich für verschiedene Ausgangssprachen Alternativsprachen definieren können, die meinem Wissen über die wahrscheinlichen Sprachkenntnisse der Nutzer entsprechen.

- möchte ich eine benutzerdefinierte Auswahl von Interface-Textbausteinen in eine Excel-Datei exportieren können, um diese einem Übersetzer zur Bearbeitung zu überlassen und später wieder zu importieren.

Aus dieser *user story* aus Perspektive eines Administrators ergeben sich die folgenden funktionalen Anforderungen:

FA201 – Bearbeiten von Übersetzungen: Das System stellt eine Administrationsoberfläche zur Verfügung, die die Bearbeitung der Übersetzungen von Interface-Textbausteinen in allen verfügbaren Sprachen ermöglicht.

FA202 – Verwaltung von Textbausteinen: Das System ermöglicht in der Administrationsoberfläche das Entfernen und Hinzufügen von Interface-Textbausteinen.

FA203 – Benachrichtigungsfunktion: Das System verfügt über Möglichkeiten, einen Administrator auf fehlende Übersetzungen hinzuweisen.

FA204 – Einpflegen neuer Textbausteine: Das System legt automatisch für unbekannte, angeforderte Textbausteine neue, leere Einträge an.

FA205 – Monitoring bestehender Textbausteine: Das System registriert es, wenn Einträge nicht länger angefordert werden, und markiert diese zur manuellen Überprüfung.

FA206 – Verschiedene Optionen für Übersetzungen: Das System verfügt in der Administrationsoberfläche je Sprache und Textbaustein über mehrere wählbare Möglichkeiten für die Übersetzung dieses Textbausteins in die jeweilige Sprache, darunter die manuelle Eintragung einer Übersetzung.

FA207 – Intelligente Vorschläge: Das System bietet in der Administrationsoberfläche durch Anbindung externer Übersetzungs-Services intelligente Vorschläge für Übersetzungen an.

FA208 – Sprachverwaltung: Das System bietet in der Administrationsoberfläche die Möglichkeit, verfügbare Sprachen hinzuzufügen und zu entfernen.

FA209 – Alternativsprachen: Das System bietet in der Administrationsoberfläche für jede Sprache die Möglichkeit, aus den verfügbaren Sprachen Alternativsprachen zu wählen.

FA210 – Excel-Export und -Import: Das System exportiert Übersetzungsdaten für Interface-Textbausteine auf eine entsprechende Anforderung aus der Administrationsoberfläche hin in eine Excel-Datei und importiert Übersetzungsdaten, wenn eine entsprechende Excel-Datei eingesendet wird.

Als Übersetzer der existierenden Anwendung...

- möchte ich neben dem Originaltext eines Interface-Textbausteins auch in einigen Fällen Angaben zum Kontext oder eine Beschreibung mitgeliefert bekommen, damit ich eine korrekte Übersetzung anfertigen kann.

Daraus ergibt sich die folgende funktionale Anforderung:

FA301 – Daten zum Kontext: Das System ordnet jedem Interface-Textbaustein einen Kontext zu, der in der Administrationsoberfläche angezeigt und beim Export mit exportiert wird.

2.1.2 Nicht-Funktionale Anforderungen

Nichtfunktionale Anforderungen machen sowohl Aussagen über die zu erfüllenden Qualitätsmerkmale des Systems, als auch über Rahmenbedingungen, die für das fertige System gelten – und insbesondere enthalten sie auch Details zu technischen Rahmenbedingungen. [1, Kapitel 11] spricht sich für eine Einteilung der nicht-funktionalen Anforderungen in Kategorien aus, die hier im Folgenden in verkürzter Form übernommen werden sollen. Dabei sollen die Kategorien je eine deutsche Bezeichnung tragen während die ursprünglich englischsprachigen Bezeichnungen in Klammern referenziert werden.

Aussehen und äußere Form (engl. *Look and Feel*)

NF101 – Valides und semantisches Markup: Das System produziert sowohl in der Administrationsoberfläche als auch in der Ausgabe übersetzter Passagen einwandfreies HTML5, das sich optimal in alle modularen Webanwendungen integrieren lässt und zu einer barrierefreien Bedienbarkeit beiträgt.

NF102 – Schlichter, moderner Stil: Das System nutzt für seine Darstellung in der Administrationsoberfläche einen schlichten Stil in Anlehnung an populäre CSS-Frameworks wie Bootstrap und ist dadurch zu vielen Webanwendungen bereits ohne Anpassungen visuell kompatibel.

NF103 – Responsivität: Das System setzt in der Administrationsoberfläche auf ein responsives Design und ist daher auf verschiedenen Endgeräten optimal darstellbar und bedienbar, was der flexiblen Einbindung als Komponente in verschiedenste Webanwendungen zugute kommt.

Bedienbarkeit und Mensch-Maschine-Interaktion (engl. *Usability and Humanity*)

NF201 – Intuitivität: Das System verfügt über eine intuitive Menüführung und kann unkompliziert bedient werden.

NF202 – Unauffällig für Anwender: Das System fügt sich aus Sicht der Nutzer der existierenden Anwendung ohne sichtbare Bruchstelle in die existierende Anwendung ein und enthält ihnen jegliche Komplexität vor.

NF203 – Zuvorkommend für Administratoren: Das System verfügt über Komfortfunktionen, die den Administratoren die Bedienung des Systems erleichtern.

Leistungsaspekte und Ressourcennutzung (engl. *Performance*)

NF301 – Verzögerungsarm: Das System lädt übersetzte Textbausteine so, dass im Laufe der Bedienung keine durch das System verursachten, zusätzlichen Verzögerungen für den Benutzer der existierenden Anwendung entstehen.

NF302 – Datensparsamkeit: Das System lädt nur so viele Daten, wie voraussichtlich auch benötigt werden.

NF303 – Optimierung: Das System bündelt je nach Situation so viele Anfragen wie möglich und reagiert dadurch auf verschiedene Situationen mit einem optimierten Verhalten, z.B. im Hinblick auf die Zahl der benötigten HTTP-Requests.

Ausführungsumgebung und Anbindung (engl. *Operational*)

NF401 – Angular: Das System soll als Angular-Anwendung realisiert werden.

NF402 – Modulare Bibliothek: Das System ist eine Angular-Bibliothek, die als Modul ohne viel Aufwand in existierende Angular-Anwendungen integriert werden kann.

NF403 – Externe Datenhaltung via REST-Schnittstelle: Das System lagert die Datenhaltung an ein externes System hinter einer REST-Schnittstelle aus, die es bedient, so dass die Datenhaltung unabhängig vom System und individuell auf die existierende Anwendung zugeschnitten konfiguriert werden kann.

Instandhaltung und Betreuung (engl. *Maintainability and Support*)

NF501 – Wartungsarm: Das System kommt ohne Wartung und Updates aus.

NF502 – Rückwärtskompatible Updates: Zukünftige Funktionsupdates für das System sind grundsätzlich rückwärtskompatibel in Funktionalität und Datenhaltung.

NF503 – Erhalt von Individualisierungen: Das System sieht gut dokumentierte Platzhalter und Stellen für Individualisierung vor, die durch ordnungsgemäße Updates nicht verändert oder zurückgesetzt werden.

Sicherheitsaspekte (engl. *Security*)

NF601 – Clientseitige Browseranwendung: Das System läuft als Angular-Anwendung im Browser des Benutzers und ist dort vor vielen denkbaren Übergriffen bereits durch die Sicherheitsmaßnahmen des Browsers geschützt.

NF602 – Eingebettet in ein bestehendes Sicherheitskonzept: Das System ist eine Komponente, die in existierende Anwendungen eingebettet wird. Als solche hat sie Anteil an der Sicherheitsinfrastruktur der übergeordneten Anwendung und muss in diese durch den jeweiligen Betreuer der existierenden Anwendung eingebettet werden.

Kulturelle und politische Aspekte (engl. *Cultural and Political*)

NF701 – Trennung von regionaler Herkunft und Sprache: Das System unterscheidet zwischen Sprache und regionaler Herkunft und vermischt sprachliche Aspekte nicht mit regionaler Symbolik. Das System benutzt regionale Aspekte für Alternativsprachen und Lokalisierung.

2.2 Testumgebung: Casa del Diavolo

Da eine Bibliothek entwickelt werden soll, die zur Internationalisierung existierender Anwendungen dient, wird sowohl im Testbetrieb während der Entwicklung als auch zu Vorführzwecken eine minimale, existierende Anwendung benötigt, in die die zu entwickelnde Bibliothek eingebettet werden kann.

Zu diesem Zweck ist es nicht zielführend, eine tatsächlich existierende Anwendung mit entsprechender Komplexität heranzuziehen, da dies Fehlerquellen und Intransparenz vermehrt. Vielmehr ist es von Vorteil, ein eigenes Minimalbeispiel zu entwerfen, das selbst keine komplexe Funktionalität zur Verfügung stellt, sondern nur in den Bereichen, die für die Funktionsdemonstrationen und -tests der Bibliothek von Bedeutung sind, tatsächlich ausgebaut ist.

Im Rahmen dieser Arbeit soll daher als minimale Testumgebung für unsere Bibliothek die Homepage eines gewissen fiktiven Pizzalieferdienst namens *Casa del Diavolo*[1], zu sehen in Abbildung 2.1, genutzt werden.

Die Homepage des *Casa del Diavolo* ist eine Angular-Anwendung, die in der Basisversion in nur einer einzigen Komponente den gesamten Homepage-Inhalt ausliefert. Sie besteht aus einer Überschrift, einem kurzen Infotext zur fiktiven Geschichte des Hauses, sowie einer Bestellmöglichkeit für verschiedene Pizzen mit entsprechenden Auswahlmöglichkeiten. Per Klick auf den Bestellen-Button wird das Bestellformular abgeschickt und die Bestellung daraufhin von einem kurzen Skript ausgelesen und noch einmal ausgegeben. Auf komplexe Logiken wurde komplett verzichtet. Auf die Responsivität des Designs der Homepage wurde insbesondere deshalb Wert gelegt, um später die

[1] Der Name „Casa del Diavolo“ entspricht in etwa dem deutschen Ausspruch „Wo der Pfeffer wächst“ in italienischer Sprache.

Einhaltung der nicht-funktionalen Anforderung **NF103** (siehe S. 11) einfach testen zu können.

Das genannte Minimalbeispiel beinhaltet viel Potenzial für den Einsatz der im Rahmen dieser Arbeit zu entwickelnden Internationalisierungs-Bibliothek. So müssen zur Internationalisierung dieser minimalen Testumgebung ganz unterschiedlich lange Texte in verschiedene Sprachen übertragen werden, wobei die Texte teilweise mit HTML-Markup durchsetzt sind und teilweise auch Kindelemente zu übersetzenden Text mitenthalten. Auch Formularelemente, die übersetzt werden müssen, sind mit vorgesehen. Damit werden alle Anwendungsfälle abgedeckt, die im Hinblick auf die Internationalisierungs-Bibliothek in einer existierenden Anwendung zu erwarten sind. Es eignet sich damit für die Überprüfung der Bibliothek auf die Erfüllung der funktionalen Anforderungen, die bereits anhand der Analyse von *user stories* gefunden wurden.

Ein Auszug aus dem HTML-Template der Basisvariante der Homepage von „Casa del Diavolo“, die noch keine bibliothekspezifischen Änderungen enthält, befindet sich im Anhang in Listing 1.1 (siehe Seite 51). Der Vergleich dieser Basisvariante mit den notwendigen Änderungen nach Implementierung und Einfügen des zu implementierenden Systems wird wichtige Aufschlüsse darüber liefern, inwiefern die nichtfunktionale Anforderung **NF402** (siehe Seite 12, im Hinblick auf Integration ohne viel Aufwand) durch das System hinreichend erfüllt wird.

Abbildung 2.1: Homepage des fiktiven Lieferdienstes *Casa del Diavolo*, bislang ohne Internationalisierung. Zu sehen ist im oberen Bereich die Überschrift mit dem Infotext und im unteren Bereich das Bestellformular. Die verwendeten Bilder sind entweder gemeinfrei oder stehen unter einer vergleichbaren Lizenz. Der Quelltext des dazugehörigen Templates ist in Listing 1.1 (S. 51) abgedruckt.

2.3 Die Grundstruktur

Wie bereits erwähnt soll hier kein vollständiger Überblick über die Konzeptionsphase, sondern nur ein Einblick in die wichtigsten Überlegungen gegeben werden. Deshalb ist es an dieser Stelle nicht zweckmäßig, Detailfragen in den Fokus zu nehmen oder gar für alle geplanten Komponenten, Klassen und Funktionen entsprechende Visualisierungen oder UML-Diagramme zu diskutieren. Stattdessen steht an dieser Stelle ein kompakter Überblick über die wesentlichen Elemente der Bibliothek im Vordergrund.

Für die Administrationsansicht wird eine Komponente benötigt, die an beliebiger Stelle eingebunden werden kann und die Bezeichnung **intman-admin** trägt. Sie muss der existierenden Anwendung direkt zur Verfügung stehen, kann in der Administrationsoberfläche der existierenden Anwendung eingebunden werden und bündelt alle Verwaltungsfunktionen, die die Bibliothek in Form einer Benutzeroberfläche zur Verfügung stellt.

Die Administrationsansicht nutzt einige weitere Komponenten, die auf die von ihnen zur Verfügung gestellten Funktionalitäten spezialisiert und zugeschnitten sind. Im Gegensatz zu *intman-admin* sollen diese Komponenten allerdings nicht direkt in der existierenden Anwendung zur Verfügung stehen, da sie nur intern verwendet werden sollen. Insbesondere soll es nicht möglich sein, die spezialisierten Komponenten selbst anzuordnen. Diese Designentscheidung schränkt die Flexibilität bei der Einbindung der Verwaltungsoberfläche geringfügig ein, da die Struktur und der Aufbau der einzelnen Bedienelemente durch die Anwendung nicht mehr individuell angepasst werden kann. Gleichzeitig wird aber dadurch auch ein großes Maß an Zuverlässigkeit (engl. *reliability*) geschaffen, da die einzelnen Komponenten besser aufeinander abgestimmt werden können; außerdem sei erwähnt, dass mit der Zahl möglicher Konfigurationen auch die Komplexität der Einbindung steigt. An dieser Stelle ist insbesondere auch die nichtfunktionale Anforderung **NF402** (siehe Seite 12) in die Abwägung einzubeziehen, die der einfachen Einbindung Vorrang gegenüber der flexiblen Einbindung gibt. Um diesen Nachteil teilweise auszugleichen sind jene internen Komponenten in einem einfachen Stil gehalten. Dabei soll gemäß **NF503** (Seite 13) optischen Anpassungen durch die Anwendung so viel Gestaltungsspielraum wie möglich einräumt werden, gleichzeitig aber die zufriedenstellende

Verwendung ohne Anpassungen nach **NF102** (Seite 11) nicht infrage gestellt werden. Die Administrationsansicht und ihre internen Unterkomponenten setzen die funktionalen Anforderungen **FA201** bis **FA203** und **FA206** bis **FA210** (Seiten 9 bis 10) um.

Die zweite wichtige Komponente, die für die direkte Verwendung in der Anwendung zur Verfügung stehen muss, ist **intman-switcher**. Dabei handelt es sich um das Element, das in der Benutzeroberfläche die Sprachauswahl und das Wechseln zwischen den verfügbaren Sprachen ermöglicht. Diese Komponente ist durch ihre Einbindung Teil der öffentlichen Benutzeroberfläche der bestehenden Anwendung – im Gegensatz zur Administrationsansicht, die im Verwaltungsbereich der bestehenden Anwendung ihren Platz findet. Während im Verwaltungsbereich ein schlichter, einheitlicher Stil sinnvoll ist, muss die Sprachauswahl dafür geeignet sein, in eine aufwändig gestaltete Oberfläche mit ganz unterschiedlichen Anforderungen eingebunden zu werden. Einerseits bedingt das wie bei der Administrationsansicht eine möglichst schlichte Gestaltung mit vielen Möglichkeiten zur Individualisierung durch die bestehende Anwendung. Andererseits soll es hier, im Gegensatz zur Administrationsansicht, auch Konfigurationsmöglichkeiten geben um die Bedienelemente des Sprachwahl-Elements flexibel an das Bedienkonzept der bestehenden Anwendung anzupassen. Dadurch wird gewährleistet, dass sich dieser Teil der Bibliothek, der sehr prominent in der bestehenden Anwendung zu sehen sein wird, gemäß **NF202** (Seite 12) ohne bemerkbare Bruchstelle in die Anwendung einfügt und das harmonische Gesamtbild für die Benutzer der Anwendung nicht negativ beeinflusst. Die Sprachwahlkomponente setzt die funktionalen Anforderungen **FA104** um.

Das Herzstück der Bibliothek ist ohne Zweifel das Element, das für das Ersetzen der Textbausteine durch die dafür konfigurierten Übersetzungen in der ausgewählten Sprache zuständig ist. Die Wahl für die konkrete Realisierung dieses Elements hat große Auswirkungen darauf, wie viel Aufwand auf Administratoren der existierenden Anwendungen bei der initialen Einbindung der Bibliothek zukommt. Gemäß **NF402** (Seite 12) ist dieser Aufwand so gering wie nur möglich zu halten. Dieses Element muss nicht nur die Mechanik zum Ersetzen der Textbausteine bereithalten, sondern auch Daten entgegennehmen, die zur Identifizierung der Textbausteine und zur Steuerung ihres Verhaltens dient.

Angular bietet mehrere Möglichkeiten, ein solches Element zu realisieren; darunter Komponenten, Direktiven oder Filter (für *pipes*). Außerdem wäre es denkbar, komplett auf eigene Elemente zu verzichten und die auszugebenden Texte sowie ihre Übersetzungen in einem Service vorzuhalten; die Einbindung der Texte erfolgt in diesem Fall dann über *interpolation binding*, wie es in [3, The Application Shell] bereits sehr früh eingeführt wird.

Die Realisierung als Komponente ist in der Implementierung besonders einfach, bringt aber viele Schwierigkeiten bei der Einbindung mit sich. So müsste jeder Textinhalt, der übersetzbar sein soll, in diese spezielle Komponente eingefasst werden. Das Template der Komponente umfasst dann zwangsläufig ein *interpolation binding*, das den durch die Komponente ersetzten Text in die Anwendung bringt. Ein besonderes Problem ergibt sich aber bei Textinhalt, der im Markup der ursprünglichen Anwendung in verschiedene HTML-Elemente verschachtelt war. Diese Verschachtelung lässt sich in einer Komponente nicht allgemeingültig abbilden, daher müsste man umgekehrt die Komponente an entsprechend vielen Stellen in der untersten Ebene der Verschachtelung einsetzen. Dadurch wird der Aufwand zur Einbindung der Bibliothek für den Administrator der bestehenden Anwendung deutlich erhöht, was weder sinnvoll ist, noch nach **NF402** gerechtfertigt werden kann. Außerdem trennt diese Maßnahme logisch zusammenhängende Textblöcke auf mehrere Instanzen der Komponente. Das ist nicht nur für die Kontextsuche bei der Übersetzung sehr problematisch, sondern schon deshalb, weil durch den unterschiedlichen Duktus verschiedener Sprachen das Markup nicht immer unverändert auf die wörtliche Übersetzung passt. Stattdessen muss es angepasst werden, die Erfüllung der funktionalen Anforderung *FA301* (Seite 10) wäre also vor weitere Hürden gestellt. Es leuchtet ein, dass es im Allgemeinen vorteilhaft ist, inhaltlich zusammenhängende Textblöcke auch in der Übersetzung als Einheit und nicht getrennt zu behandeln. Die Möglichkeit der Realisierung als Komponente sowie die Möglichkeit der Bereitstellung über einen Service und Einbindung über *interpolation binding* ist aus diesen sachlichen Gründen daher zu verwerfen.

Auch die Möglichkeit zur Realisierung als Filter zur Verwendung mit *pipes* hat einige dieser Schwächen; allem voran muss auch hier eine Ersetzung auf der untersten Ebene im Markup erfolgen. Insofern zeigt die Realisierung mit Filter eine große Ähnlichkeit

zur Realisierung via *interpolation binding*. Auf den ersten Blick scheint es so, dass die Zielsetzung, einen gegebenen Text zu übersetzen, sehr gut auf die Charakteristika von Filter und *pipes* passt. Dem ist aber auf den zweiten Blick nicht so. Für gewöhnlich werden Filter und *pipes* dafür verwendet, um variable Daten in einem fest vorgegebenen Format anzuzeigen. Die Anforderung hier ist umgekehrt, denn hier gibt es statische Daten (je nach Implementierung ein Text in Originalsprache oder ein gewisser Identifier), die dann in einem variablen Format (konkret in einer variablen Sprache) ausgegeben wird! Die Charakteristika der Funktionalität unterscheiden sich also bei genauerem Hinsehen in wesentlichen Punkten, was die Argumente für eine Realisierung via *pipe* durchaus schmälert.

Eine sinnvolle Lösung ist die Realisierung als Direktive, genauer als *attribute directive*. Direktiven haben den Vorteil, dass sie ohne größeren Aufwand in Templates integriert werden können. Direktiven können – vergleichbar mit HTML-Attributen – grundsätzlich an jedes im bestehenden Template vorkommende HTML-Element und jede dort verwendete Komponente angehängt werden. Das ermöglicht es auch, tief geschachtelte Strukturen mit einer einzelnen Direktive am beinhaltenden Element für die Übersetzung zu markieren. Durch eine geschickte Implementierung ist es dann auch grundsätzlich möglich, einem Übersetzer direkt den notwendigen Kontext zu liefern: Wenn gleich ganze zusammenhängende Textabschnitte in Einem übersetzt werden, erübrigt sich die zusätzliche Verwaltung von Informationen zum Kontext. Natürlich muss eine gewisse Trennung beibehalten werden, um später die Zuordnung der übersetzten Textabschnitte zu den verschiedenen Ebenen der Verschachtelung zu ermöglichen. Dies kann aber auch dadurch geschehen, dass die Übersetzungsdaten, die einer Direktive zugeordnet werden, in voneinander getrennte Unterabschnitte unterteilt werden. Diese Lösung hält den Einrichtungsaufwand für den Administrator der existierenden Anwendung minimal, bietet gleichzeitig einen sinnvollen Umgang mit verschachtelten Strukturen und erübrigt die Frage nach einem Kontext für den Übersetzer. Zusätzlich können durch automatische Erzeugung einer ID für die genannte Direktive sogar durch strukturelle Direktiven automatisch erzeugte Elemente mit einer Übersetzung versorgt werden. Gemäß dieser Zielsetzung trägt die Direktive die Bezeichnung **intmanId** und beinhaltet als Wert einen individuellen Bezeichner für den Textblock. Die Realisierung als Direktive sorgt im Üb-

rigen auch dafür, dass die Anforderung **NF101** (Seite 11) grundsätzlich bereits erfüllt ist, sofern die existierende Anwendung sie ebenfalls erfüllt. Neben *intman-admin* und *intman-switcher* handelt es sich bei der Direktive *intmanId* um die dritte und letzte Struktur, die der einbindenden Anwendung direkt zur Verfügung gestellt werden muss. Alle anderen Teile der Bibliothek bleiben der internen Verwendung der Bibliothek vorbehalten und dürfen nicht durch die Anwendung selbst aufgerufen oder eingesetzt werden.

Gemäß [3, Services] gehört es in Angular zum guten Stil, die durch Komponenten und Direktiven zur Verfügung gestellte Funktionalität möglichst schlank und in starkem Bezug zur tatsächlichen Benutzerinteraktion zu halten. Daher erscheint es zweckmäßig, die komplexeren Funktionalitäten, die für das erfolgreiche Austauschen von Inhalten durch übersetzte Texte benötigt werden, in eine andere Struktur auszulagern und diese dann mit der genannten Direktive zu verknüpfen. Eine Möglichkeit wäre an dieser Stelle ein Service, der auf Ebene der einzelnen Elemente gebunden ist und dann in entsprechend vielen Instanzen vorkommt; die Erstellung wäre dann via *dependency injection* an den anwendungsweiten Injector gebunden. Das sehr mächtige Konzept der *dependency injection* ist in Angular weit verbreitet und gut unterstützt, bedeutet aber auch eine gewisse Komplexität. Es kommt dazu, dass die Übersetzungen und ihr Management über einen anwendungsweiten Service erfolgt, der dann wiederum jeweils injiziert werden müsste. Da für den speziellen Anwendungszweck nur wenig konkrete Vorteile für die Realisierung via Service sprechen, soll die Funktionalität an dieser Stelle stattdessen durch zwei Klassen gekapselt werden, die der oben angesprochenen Unterteilung der logisch zusammengehörigen Textblöcke Rechnung tragen und in der konkreten Handhabung zwar weniger mächtig, aber auch weniger komplex sind.

Die Klasse **TextContainer** repräsentiert dabei den logisch zusammenhängenden Textblock, der mit einer eigenen *intmanId*-Direktive ausgezeichnet wird. Der *TextContainer* wird dabei bei der Initialisierung der Attributdirektive erzeugt und nimmt bis zum Ende ihres *lifecycle* alle Aufgaben der Verwaltung von Texten und Übersetzungen wahr. Die Attributdirektive selbst dient daher nur zur Kennzeichnung der betreffenden Stellen im *template*, die Funktionalität und Verwaltung der Übersetzungsdaten werden durch die der Direktive zugeordneten Instanz der Klasse *TextContainer* bereitgestellt. Der *TextContainer* verwaltet auch die interne Datenrepräsentation des logisch zusammenhängenden

Textblocks und macht diese der zentralen Infrastruktur in ihrer jeweils aktuellen Form zugänglich. Die Klasse setzt dabei die funktionalen Anforderungen **FA102** (Seite 7) und **FA103** (Seite 8) sowie **FA203** (Seite 9) um.

Die Repräsentation der einzelnen Textelemente eines *TextContainer*, voneinander durch verschiedene Verschachtelungstiefen im *template* getrennt, stellt die Klasse **TextualContent** dar. Dabei analysiert der *TextContainer* im Zuge seiner Initialisierung in dem nativen Element, welches ihm durch die Direktive zugeordnet wurde, die vorliegende Struktur des DOM. Zu jeder Textstelle innerhalb der Verschachtelung erzeugt er dabei eine Instanz von *TextualContent*, die die Übersetzungsfunktionen für die einzelne Textstelle bereitstellt. Sie kommuniziert ihrerseits wieder mit dem übergeordneten *TextContainer* bezüglich des Status der einzelnen Textstelle. Die Übersetzungen, die angefordert werden müssen, werden nicht direkt in der zentralen Infrastruktur, sondern zunächst im *TextContainer* angefordert. Das ermöglicht eine Optimierung des Anfrageverhaltens durch den *TextContainer*, wie sie in **NF303** (Seite 12) gefordert wird. Die einzelnen Instanzen von *TextualContent* sind dafür zuständig, die Übersetzung in der Standardsprache aus dem initialen Code der Anwendung zu entnehmen und der zentralen Infrastruktur zugänglich zu machen. Dem gemäß werden durch diese Klasse die Anforderungen **FA101** (Seite 7) und **FA102** (Seite 7) erfüllt.

Die genannte zentrale Infrastruktur wird, wie in Angular üblich, durch einen anwendungsweiten *service* bereitgestellt, den **IntmanLibService**. Dieser *service* stellt verschiedene Helfer-Funktionalitäten bereit:

1. **Kommunikation mit dem Server:** Die von der Bibliothek benötigten Daten werden gemäß **NF403** (Seite 13) über eine REST-Schnittstelle bereitstellt. Dazu wird ein *HttpClient* genutzt, der durch die einbindende Anwendung über *dependency injection* bereitgestellt wird. Dies ermöglicht auch – wie in [3] ausgeführt – für die Zeit der Entwicklung die *HttpClientInMemoryWebApi* zur Simulation der REST-Schnittstelle zu nutzen, ohne, dass die Bibliothek zum produktiven Einsatz noch verändert werden muss, da der genutzte *HttpClient* nur durch die einbindende Anwendung kontrolliert wird.

2. **Registrierung von Komponenten und Klassen der Bibliothek:** Damit bei einer Änderung von Übersetzungsdaten die betroffenen Teile der Bibliothek über die Änderung benachrichtigt und zu einer Aktualisierung ihrer *view* angeregt werden können, müssen diese zunächst zentral registriert werden. Diese Funktionalität ist insbesondere auch eine Umsetzung der Anforderung **NF203** (Seite 12), damit Administratoren von ihnen vorgenommene Änderungen direkt beobachten können. Zwar hat Angular über *interpolation binding* und *two way data binding* bereits mächtige Werkzeuge, um derartiges automatisiert zu erledigen; wir hatten uns aber aus genannten Gründen für eine Lösung ohne *data binding* entschieden und müssen daher diese Funktionalität für die als Lösung gewählte Direktivenrealisierung selbst bereitstellen.
3. **Anwendungsweite Sprachverwaltung:** *IntManLibService* hält die ID der Standardsprache abrufbereit, hält die Information über die aktuell ausgewählte Sprache bereit und löst bei einem Wechsel der Sprache über *intman-switcher* in allen registrierten Instanzen von *TextContainer* den Sprachwechsel aus.

Es gibt unterschiedliche Möglichkeiten, Angular-Strukturen abzubilden, wie [4] ausführt. Abbildung 2.2 visualisiert die Beziehungen der genannten wichtigen Elemente untereinander in Form eines aus Gründen der Übersichtlichkeit stark reduzierten UML2-Klassendiagramms, wobei die Angular-spezifischen Strukturen wie *Services*, *Components* und *Directives* vereinfacht als Klassen dargestellt sind.

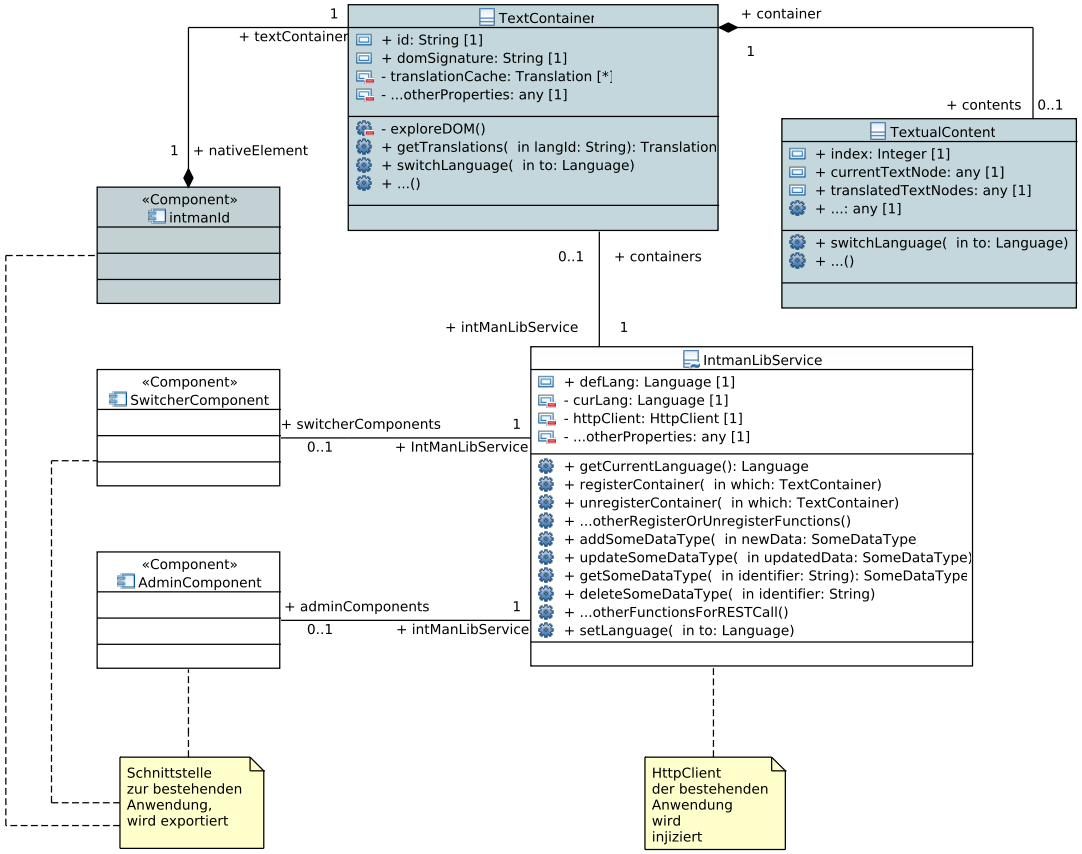

Abbildung 2.2: Klassendiagramm zur Grundstruktur der Bibliothek und zur Visualisierung der Beziehungen zwischen den wichtigsten Elementen. Es handelt sich um ein stark reduziertes Klassendiagramm nach dem UML2-Standard, soweit dieser auf Angular-Strukturen angewandt werden kann. Im oberen Teil der Abbildung finden sich die Elemente, die für die Ersetzung der Texte notwendig sind: Die Klassen *TextContainer* und *TextualContent* sowie die Direktive *intmanId*. Die Elemente auf der linken Seite stehen der einbindenden Anwendung direkt zur Verfügung. In der Mitte steht der zentrale Service, der alle Funktionen bereitstellt - darunter auch den Zugriff auf die REST-Schnittstelle mit den benötigten Daten mittels eines aus der bestehenden Anwendung injizierten *HttpService*. Die Anzahl der aufgeführten Funktionen und Eigenschaften ist hier auf das für die Grundstruktur wesentliche Maß beschränkt.

2.4 Umgang mit Alternativsprachen

Die funktionalen Anforderungen **FA209** (Seite 10 - Alternativsprachen), **FA102** (Seite 7 - Ausliefern von Textbausteinen in alternativer Sprache) sowie **FA103** (Seite 8 - Angemessene Wahl einer alternativen Sprache) stellen eine gewisse Herausforderung dar, da die Wahl einer alternativen Sprache im Zweifel nicht trivial ist.

Um diese Problematik aufzulösen, muss zunächst geklärt werden, welche Kriterien eine alternative Sprache erfüllen muss und wie Sprachen überhaupt kategorisiert werden können. Dabei kann man als Anhaltpunkt auf internationale Vereinbarungen und Standards zurückgreifen. Für die Standardisierung im World Wide Web ist das W3C[2] zuständig. In der Specification zum *living standard* HTML5 ist festgelegt, dass Sprachangaben in HTML, zum Beispiel im *lang*-Attribut, dem Standard **BCP47** (siehe [5]) folgen müssen. Es ist daher naheliegend, auch im vorliegenden Fall die Sprach-ID nach diesem Muster festzulegen. *BCP47* löst dabei die Bestimmungen von **RFC1766** (siehe [6]) ab und enthält insbesondere auch die Bestimmungen in RFC1766 als Teilmenge, so dass für die Zwecke der Bibliothek RFC1766 als Grundlage gewählt werden kann ohne BCP47 zu verletzen.

Gemäß [6] setzt sich ein Sprachbezeichner aus (mindestens) zwei Teilen zusammen, die untereinander durch einen Bindestrich getrennt sind. Der erste Teil besteht außer in Ausnahmefällen aus zwei Kleinbuchstaben, die für die Sprache stehen. Die Spezifikation in [6, The Language Tag] kennt dabei genau zwei Ausnahmefälle:

1. Bei Sprach-Tags, die mit `x-` beginnen, handelt es sich um zu bestimmten Zwecken selbst vergebene Tags, deren restlicher Inhalt auch nur selbst bestimmten Regeln folgt; es ist dort also keine allgemeingültige (d.h. andwendungsübergreifende) Interpretation möglich und die gültigen Bezeichner werden auch nicht zentral registriert.

2. Sprach-Tags, die mit `i-` beginnen, kennzeichnen standardisierte Sprachbezeichnungen, die durch die IANA registriert wurden, aber kein eigenes sprachspezifisches Zwei-Buchstaben-Kürzel besitzen, da es sich beispielsweise um eine

[2] World Wide Web Consortium, `https://www.w3.org/`

exotische Kunstsprache oder um eine nur lokal verbreitete Sprache handelt. [5, Primary Language Subtag] nennt an der Stelle auch zwei Beispiele: die klingonische Sprache (Kunstsprache aus Star Trek, `i-klingon`) und Bunun (Sprache eines indigenen Volks auf Taiwan, `i-bnn`).

Der zweite Teil des Sprach-Tags enthält zusätzliche Informationen über die Sprachvariante. Dies kann ein Ländercode nach **ISO 639** in Großbuchstaben sein oder Bezeichnungen für Dialekt- oder Schriftvarianten.

So werden beispielsweise die vier Amtssprachen der Schweiz durch folgende Sprach-Tags gekennzeichnet:

- `de-CH` – Schweizer Hochdeutsch
- `fr-CH` – Schweizer Französisch
- `it-CH` – Schweizer Italienisch
- `rm-CH` – Rätoromanisch

Es sind an diesem Beispiel nun drei verschiedene Arten erkennbar, eine Alternativsprache sinnvoll zu wählen:

1. Die Standardsprache – um zu gewährleisten, dass auch bei vollständig fehlenden Übersetzungen kein Textbaustein ohne Inhalt vorkommt, muss die Auslieferung eines Textes in der Standardsprache der Anwendung immer eine mögliche Alternative sein, da sich die Standardsprache zur Beschaffung ihrer Textbausteine direkt im Quelltext der bestehenden Anwendung bedient und daher die einzige Sprache darstellt, bei der garantiert werden kann, dass eine Übersetzung vorliegt. Wenn eine automatische Ersetzung durch einen standardsprachlichen Baustein erfolgt, muss der Benutzer zwingend darüber informiert werden, dass für den Textblock leider keine Übersetzung vorliegt, damit er den sprachlichen Bruch innerhalb der Anwendung versteht und nötigenfalls gemäß **FA104** (Seite 8) auf eine andere Sprache wechseln kann.
2. Eine andere Variation der selben Sprache – es kann vorkommen, dass einzelne Textbausteine zwar in Deutsch (`de-DE`), aber nicht in Schweizer Hochdeutsch

(`de-CH`) vorliegen. Dann ist es sinnvoll, die fehlenden Passagen aus der Übersetzung nach `de-DE` zu entnehmen, da die sprachlichen Unterschiede unter verschiedenen Sprachvariationen keinen so großen Bruch bedeuten, wie er durch einen Wechsel der kompletten Passage in eine andere Sprache entstehen würde. Beispielsweise mag für Bundesdeutsche der „1. Januar 1999“ intuitiver sein als das österreichisch-deutsche Pendant „1. Jänner 1999“, es handelt sich aber immer noch um eine eher niederschwellige Alternative. In vielen Fällen besteht zwischen den Sprachvariationen sogar noch weniger Unterschied als bei diesem Beispiel. Um keine unnötigen Brüche in der Benutzung der Anwendung zu erzeugen sollte der Benutzer über eine solche alternative Sprachwahl auch nicht extra informiert werden.

3. Eine andere Sprache derselben geografischen Region – man kann annehmen, dass ein Schweizer, der das Schweizer Hochdeutsch wählt, auch andere Amtssprachen der Schweiz versteht. Daher könnte eine sinnvolle Alternativsprache auch aufgrund des Ländercodes ausgewählt werden. Dem steht allerdings gegenüber, dass anders als bei einer Variation der selben Sprache wieder die gesamte Passage ersetzt werden muss, da die Unterschiede zwischen verschiedenen Sprachen einer geografischen Region erheblich größer sind als die Unterschiede zwischen verschiedenen Variationen der selben Sprache. Gleichwohl müssten die Benutzer informiert werden. Sehr problematisch ist an dieser Stelle die Wahl der konkreten Alternative. Die Bibliothek kann die Frage, ob ein Benutzer, der Schweizer Hochdeutsch wählt, Rätoromanisch, Französisch oder Italienisch besser versteht, nicht beantworten. Eine zufällige Wahl aus diesen möglichen Alternativen ist nicht zu vermitteln – auch vor dem Hintergrund, dass man nicht davon ausgehen kann, dass tatsächlich alle Personen, die Schweizer Hochdeutsch wählen würden, auch die restlichen Amtssprachen der Schweiz verstehen.

Die ersten beiden Möglichkeiten können durch die Bibliothek implementiert und automatisiert werden. Bei der Ersetzung durch die Standardsprache ist nichts weiter zu beachten, bei der Ersetzung durch eine andere Sprachvariation kann man sich die Eigenheiten des RFC1766 zunutze machen: sofern der Teil des Sprachtags, der vor dem Bindestrich

liegt, genau zwei Buchstaben umfasst, handelt es sich bei allen Sprachen, deren Tag mit den selben zwei Buchstaben beginnt, um mögliche Alternativsprachen.

Der dritten Möglichkeit zur Ersetzung muss eine Benutzerentscheidung zugrunde liegen, daher spricht vieles dafür, diese Alternativmöglichkeit nicht weiter auszugestalten. Es ist an der Stelle ausreichend, wenn dem Benutzer angezeigt wird, dass keine Übersetzung in der gewählten Sprache vorliegt, damit der Benutzer über das normale Sprachwahlelement eine andere Sprache auswählen kann, sofern er die standardsprachliche Alternative nicht versteht. Aufgrund dieser Designentscheidung ist die funktionale Anforderung **FA104** (Seite 8) bereits erfüllt, wenn die manuelle Sprachwahl über die *Switcher*-Komponente gewährleistet wird. Die Einschränkung der automatischen Wahl einer alternativen Sprache auf Variationen der selben Sprache gewährleistet darüber hinaus die Angemessenheit der Wahl einer alternativen Sprache und erfüllt damit **FA103** (Seite 8).

3

Implementierung und Dokumentation

Im folgenden Kapitel sollen nun wichtige Details der konkreten Implementierung der Bibliothek vorgestellt werden, die als Prototyp bzw. *proof of concept* angefertigt wurde. Die Quelltexte des Prototypen, auf die sich die Ausführungen dieses Kapitels beziehen, finden sich im Anhang unter A. Soweit in den Ausführungen Codezeilen als Referenz genannt werden, beziehen sich diese stets auf die dortigen Zeilennummern. Wir werden im Folgenden die einzelnen Elemente betrachten, aus der sich die Bibliothek zusammensetzt und dabei im Besonderen auf Umstände eingehen, denen bei der Realisierung großes Augenmerk gewidmet wurde, sowie auf Schwierigkeiten, die im Vorhinein nicht ohne weiteres ersichtlich waren und sich im Laufe der Implementierung ergeben haben. Es soll im Rahmen dieses Kapitels insbesondere auch an geeigneten Stellen die Gelegenheit zum Blick über den Tellerrand genutzt werden, wenn es um Abwägungen zwischen verschiedenen Realisierungsmöglichkeiten geht.

3.1 Das Grundgerüst der Bibliothek

Den Ausführungen in [7] folgend muss zunächst in Ergänzung zum Minimalbeispiel mit Hilfe des CLI-Werkzeugs eine Bibliothek im selben Angular-Projekt generiert werden. Die automatisierte Generierung einer Bibliothek ist im Standardumfang von Angular CLI seit dem Release von Version 6 im Mai 2018 möglich (siehe [8]). Diese Neuerung führte zu einer starken Vereinfachung des Entwicklungsprozesses von Bibliotheken und macht die kompliziertere manuelle Erzeugung und Strukturierung wie sie [9] vorstellt, unnötig.

Anschließend können dank der Neuerungen in Angular CLI die benötigten Komponenten, Klassen und Direktiven direkt in der Bibliothek generiert werden. Für die Komponen-

ten *intman-switcher* und *intman-admin* sowie die Direktive *intmanId*, die der existierenden Anwendung direkt zur Verfügung stehen, gilt, dass sie zusätzlich noch exportiert und der einbindenden Anwendung dadurch bekannt gemacht werden müssen. Dies geschieht an zwei Stellen: dem `ngModule` der Bibliothek (*IntManModule*) und in der *public_api*. Die zugehörigen Listings 2.1 und 2.2 (Seite 53ff.) zeigen diesen Export. Im Vergleich dazu werden die Komponenten `AdminLanguageComponent`, `AdminTranslationComponent` nur deklariert, da auf sie nicht von außerhalb des Moduls zugegriffen werden soll.

Wie bereits in Abschnitt 2.3 ausgeführt, werden für die Basisfunktionalität neben den genannten Komponenten noch die Klassen `TextContainer` und `TextualContent` benötigt. Auch sie werden durch Angular CLI in der Bibliothek generiert und ergeben später das Bild, das in Listing 2.15 bzw. 2.16 (Seite 80ff.) zu sehen ist.

3.2 Rückschlüsse auf die REST-Schnittstelle

Der `InMemoryDataService` unter Listing 3.1 (Seite 103) ist Teil der Testumgebung *Casa del Diavolo*. Wie Listing 3.2 (Seite 105) zeigt, liefert dieser Service Daten, die der Bibliothek daraufhin über eine *dependency injection* so zur Verfügung gestellt werden, als würde es sich dabei tatsächlich um via HTTP übertragene Daten handeln. Zuständig dafür ist das Modul *HttpClientInMemoryWebApi*, das in der Testumgebung eingebunden wurde. Die Bibliothek bekommt von diesem Aufbau nichts mit - sie behandelt den injizierten Service wie einen ganz normalen *HttpClient*.

Für die Dokumentation der Bibliothek ist der `InMemoryDataService` trotzdem von großer Bedeutung, da er Rückschlüsse darauf zulässt, wie eine tatsächliche REST-API, die Daten von der Bibliothek entgegen nehmen muss, auszusehen hat. Dazu können wir den Aufbau der initial statischen Datenbank in Listing 3.1 (Seite 103) analysieren und unsere Erkenntnisse auf eine REST-Schnittstelle übertragen.

In `/defLangId` wird ein String mit dem Tag der Standardsprache vorgehalten. Es handelt sich um eine serverseitige Grundeinstellung, die durch die Bibliothek nur gelesen wird – die API muss hier also nur GET-Requests erlauben. Unter `/languages` liegen Da-

ten vom Typ *Language* (vergleiche Listing 2.4 auf Seite 55), unter `/translations` liegen Daten vom Typ *Translation* (vergleiche Listing 2.5 auf Seite 56) und unter `/containerSettings` liegen Daten vom Typ *containerSetting* (vergleiche Listing 2.3 auf Seite 55). Die REST-API muss unter diesen drei Pfaden die entsprechenden Daten vorhalten, wahlweise als Array unter der genannten URL, als gefiltertes Array durch Angabe eines entsprechenden Query oder als einzelnem Wert durch Aufruf der ID unterhalb dieses Pfades. Alle drei Pfade und ihre Unterpfade können mit den HTTP-Methoden GET, POST, DELETE oder UPDATE aufgerufen werden, die REST-API muss das gewährleisten. Letzteres wird bei einer genauen Betrachtung von Listing 2.17 (Seite 94ff.) klar.

Die Authentifizierung und Autorisierung der Requests muss im Produktiveinsatz außerhalb der Bibliothek gewährleistet werden, da das natürlich nicht allgemeingültig durch die Bibliothek übernommen werden kann – es sei an dieser Stelle auch auf die nicht-funktionale Anforderung **NF602** (Seite 13) verwiesen. Dies wird durch den oben erläuterten Ansatz einer *dependency injection* optimal möglich, denn so wie in der Testumgebung ein simulierter HttpClient in die Bibliothek durchgereicht werden kann, so kann im Produktivbetrieb selbstverständlich ein richtiger HttpClient der existierenden Anwendung durchgereicht werden, welcher sich bereits im Rahmen der existierenden Anwendung individuell authentisiert hat – beispielsweise in einem Verfahren, wie von [10] beschrieben. Die Bibliothek fügt sich dort also nahtlos in das bestehende Sicherheitskonzept der existierenden Anwendung ein.

3.3 IntmanSwitcher in variabler Ausgestaltung

Die Sprachwechselkomponente `SwitcherComponent` kann durch Verwendung des Elements `intman-switcher` an jeder Stelle in der Anwendung integriert werden. Eine beispielhafte Einbindung ist in Listing 4.1 auf Seite 106 in Zeile 8 zu sehen. Dort fällt die Eigenschaft `variant` auf. Im Template von `SwitcherComponent` unter Listing 2.6 (Seite 56) wird deutlich, dass dieses Attribut dazu dient, die Funktionsweise des Sprachwahlelements zu steuern. Die verschiedenen Möglichkeiten sind in Abbildung 3.1 einander gegenübergestellt.

Wie ein Blick in die Zeilen 2, 10, 19 und 26 des Template-Codes verrät, ist das auch kein Wunder: Das Element, das den Beschreibungstext „Bitte wählen Sie Ihre Sprache:“ beinhaltet, verfügt über eine `intmanId` mit dem Wert *IntManInternalChooseLanguage*! Es handelt sich also bereits dort, innerhalb des Template von `SwitcherComponent` um eine Übersetzung, die von der Bibliothek selbst übernommen wird.

Dieses Vorgehen ist einer gewissen Optimierung zu verdanken: Um das System mit überschaubarem Aufwand verwaltbar zu machen, sollten so wenig Daten wie möglich beim Datensatz der Sprache verankert sein. Listing 2.4 (Seite 55) zeigt, dass der Sprache in ihrem Datensatz genau ein Ausdruck zugeordnet ist: Der, der den Benutzer auf eine fehlende Übersetzung hinweist. Denn natürlich muss genau das auch gewährleistet sein: dass der Hinweis auf eine fehlende Übersetzung nicht selbst fehlt, wenn die Sprache anwählbar ist. Die Aufforderung zur Sprachwahl hingegen ist weit weniger wichtig. Man bedenke nur einmal, dass derjenige, der die Sprache wechseln möchte, selbige Aufforderung im Zweifel sowieso nicht versteht – denn sie ist ja gerade in jener Sprache verfasst, von der er mittels seiner Bedienhandlung weg möchte, da er sie nicht gut genug versteht. Vor diesem Hintergrund ist es also nicht sinnvoll, die Beschriftung fester zu verankern und umso sinnvoller, sie direkt in die Abläufe der Bibliothek mit einzubeziehen.

Die aufgeworfene Problematik damit, dass die Menschen, die die Sprache wechseln wollen, die Beschriftung zu den Bedienelementen gerade nicht lesen können, ist in Fachkreisen ein altbekanntes Problem. So beschreibt auch das W3C im Artikel [11] die Problematik, dass gerade bei den auf den ersten Blick so gut geeigneten Dropdown-Listen ein echtes Problem darin bestehen kann, die Sprachauswahl überhaupt erst als

Abbildung 3.1: Die verschiedenen Darstellungsarten von SwitcherComponent in einer Gegenüberstellung. Die Werte für `variant` sind entsprechend – beginnend rechts oben im Uhrzeigersinn – `radio`, `buttons`, `select` und `dropdown`. Besonderes Augenmerk verdient dabei die Realisierung des Texts am Sprachwechsler.

Sprachauswahl zu erkennen, wenn man der aktuell gewählten Sprache (und manchmal sogar deren Schrift) nicht mächtig ist. Demzufolge wären von den gegebenen Varianten nur die Varianten *radio* und *buttons* empfehlenswert. Eine Realisierung mit Links wie im Artikel vorgeschlagen wäre zwar theoretisch denkbar, ist aber mit der Architektur der Bibliothek nicht vereinbar - denn die Single-Page-Charakteristik einer Angular-Anwendung und die Art, in der die Bibliothek in die Anwendung eingebunden werden, erschweren das Anbieten von URL-bedingter Sprachwahl. Hierzu müsste die bestehende Anwendung ein *Router*-Modul besitzen, das je nach URL der Bibliothek via einer Direktive oder Komponente die ausgewählte Sprache zugänglich macht. So eine Information könnte sogar direkt über die Eigenschaft `lang` einer Switcher-Komponente mitgegeben werden - dann müsste aber zunächst ausgeschlossen werden, dass sich die Angaben an mehreren Switcher-Komponenten unterscheiden. Eine solche Funktionalität ist momentan allerdings nicht implementiert und entfernt sich auch verhältnismäßig weit von der Funktionsweise der bisher implementierten Varianten.

Der Übersetzer des genannten Artikels ins Deutsche erläutert in [12] ausführlich, warum solche Sprachwechsler außerdem – trotz der weiten Verbreitung – auch nicht mit Flaggensymbolen arbeiten sollten. Dass das in unserem Fall gar nicht funktionieren würde, liegt an der vorausschauenden Wahl der Sprach-Tags nach dem im World Wide Web geltenden Standard, wie sie in Abschnitt 2.4 getroffen wurde, denn schon beim Beispiel in jenem Abschnitt wird das Problem offensichtlich: Wenn jetzt dem Schweizer Hochdeutsch *de-CH* bereits die Schweizer Flagge zugeordnet wäre, wie sollte man das dann von der Schweizer Flagge unterscheiden, die für das Rätoromanische *rm-CH* steht? Deshalb wurde im Rahmen dieser Software mit einer bewussten Entscheidung auf Flaggen als Sprachwahlelement verzichtet und damit auch die nicht-funktionale Anforderung **NF701** (Seite 13) erfüllt.

3.4 Verwaltung mit AdminComponent

Noch unkomplizierter als die Einbindung von `IntmanSwitcher` ist die Einbindung von `AdminComponent`, die in Listing 4.1 auf Seite 106 in Zeile 32. Das Template der

Komponente in Listing 2.9 (Seite 63) gibt Aufschluss darüber, wie sich die Administrationsansicht zusammensetzt.

Im obersten Bereich sind demnach Hinweise zu fehlenden Übersetzungen zu finden. Ein Blick in den Quelltext der Komponente unter Listing 2.8 (Seite 59) offenbart, dass die Komponente diese Information mit einer eigenen Methode `collectMissingTranslations` ab Zeile 38 selbst zusammenträgt. Diese Methode muss einiges an Aufwand betreiben, denn bedingt dadurch, dass Sprachen und Container-Einstellungen sowie die Standardsprache asynchron vom Server geladen werden, sind hierzu einige gestaffelte Requests nötig, so in den Zeilen 40, 41 und 45. Daraufhin müssen die geladenen Daten erst umsortiert und miteinander verglichen werden, um schließlich auf die nicht-vorhandenen Daten schließen zu können. Diese Vorgehensweise erscheint umständlich und lässt vermuten, dass ein konzeptioneller Fehler bei der Datenhaltung vorliegt, der diese umständliche Herangehensweise verursacht. Es ist allerdings so, dass die Datenhaltung ja gerade nicht für die Administrationsansicht optimiert werden muss, sondern für die Benutzerseite. Es ist im Hinblick auf die insgesamt anfallende Menge von Requests deutlich wichtiger, die Anzahl der Requests auf Benutzerseite gering zu halten, da dort viel mehr Seitenaufrufe erfolgen als in der Administrationsansicht – und die Administrationsansicht hat in dem Fall leider durch die Suche nach fehlenden Übersetzungen den im Vergleich zu den häufigeren Seitenaufrufen genau umgekehrten Anspruch und unterliegt daher klar in der Priorität.

Der zweite Bereich im Template von `AdminComponent` stellt die Sprachverwaltung dar; hier kommt in Zeile 27 die das Element `intman-admin-language` vor, über das noch im Abschnitt 3.5 berichtet wird. Auf die Iteration über alle Sprachen folgt an der Stelle das Formular zum Anlegen einer neuen Sprache, das dem Formular zum Verändern bestehender Sprachen (nämlich jener `intman-admin-language`-Komponente) nachempfunden ist. Hier muss ein zu [6] kompatibler Sprachbezeichner angegeben werden, der im Nachhinein nicht mehr verändert werden kann. Sollte die Eingabe den Vorgaben nicht entsprechen, wird der Button zum Speichern, der nur bei Änderungen (`changedSomethingOnNewLang`) überhaupt sichtbar wird, wieder verborgen und stattdessen via `!newLangMeetsRequirements` in Zeile 34 ein Hinweistext eingeblendet.

Der dritte und vierte Bereich im Template enthält jeweils eine bestimmte Menge von Elementen mit Bezeichner `intman-admin-translation`, die in Abschnitt 3.6 ausführlicher diskutiert werden. Beide Bereiche liefern den Elementen je eine Sprache und einen Container, dabei unterscheidet sich allerdings der Fixpunkt in der Menge. So ist im dritten Bereich der Textblock fest vorgegeben und es werden dazu alle Übersetzungen in den verschiedenen Sprachen aufgelistet, während im vierten Bereich die Sprache fest ist und über die verschiedenen Container iteriert wird. Die Fixpunkte werden dabei durch dropdown-Auswahllisten realisiert, die die Eigenschaften `langId` und `containerId` aus Listing 2.8 (Seite 59) bestimmen.

Abbildung 3.2 zeigt eine Detailansicht aller vier Bereiche, die über das *details-summary*-Elementpaar aus dem Standard HTML5 einzeln ein- und ausgeblendet werden können.

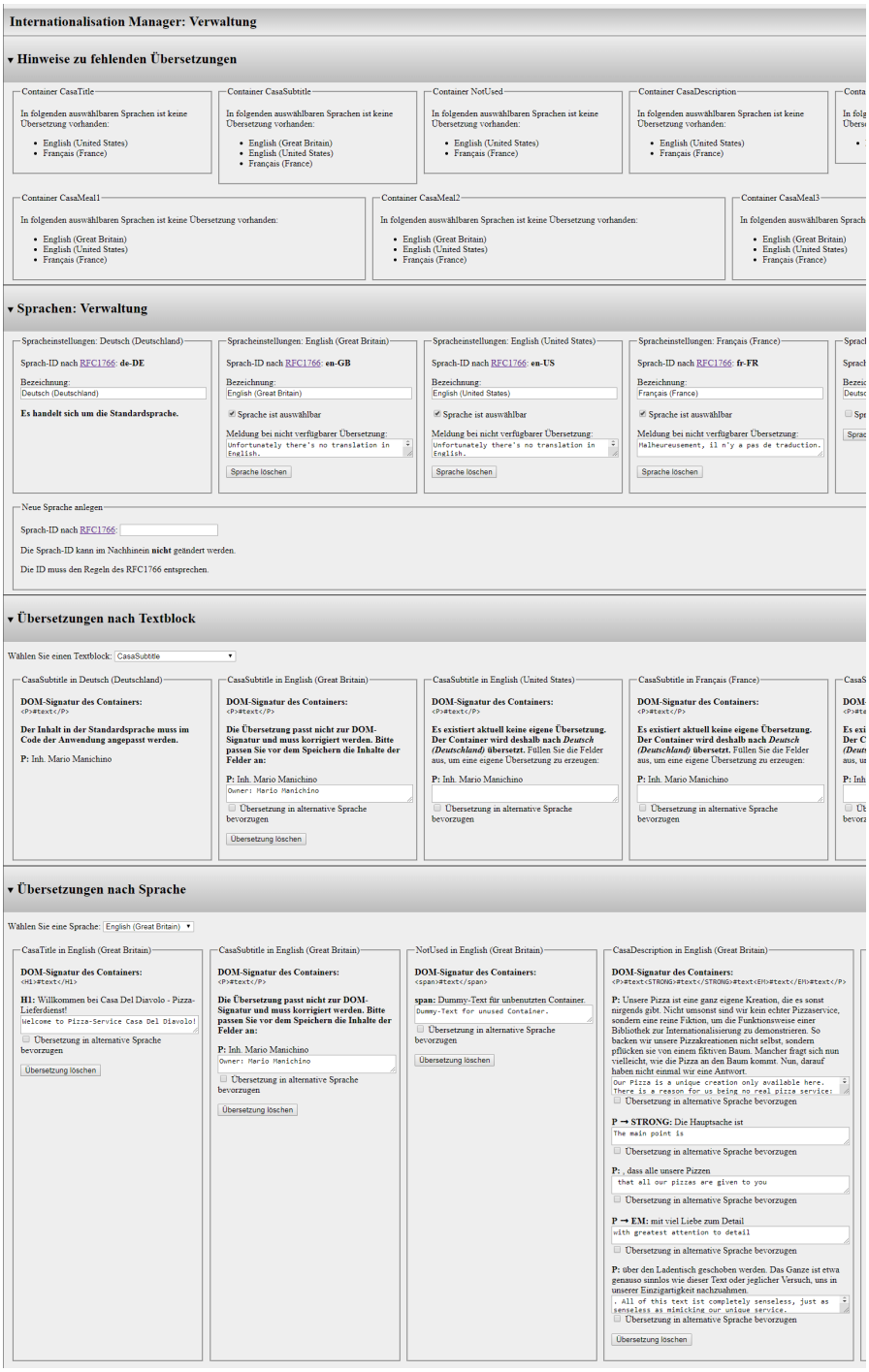

Abbildung 3.2: Gesamtansicht der `AdminComponent` bei komplett ausgeklappten Verwaltungsbereichen. Ganz oben sind die noch fehlenden Übersetzungen nach Container aufgelistet, darunter folgt die Sprachverwaltung und in der unteren Hälfte sind die beiden Herangehensweisen an das Editieren von Übersetzungen zu sehen, die das System ermöglicht.

3.5 Sprachverwaltung mit AdminLanguageComponent

In Abbildung 3.3 ist die Komponente `AdminLanguageComponent` aus Listing 2.11 (Seite 68) im Detail zu sehen. Die Aufgaben der Komponente selbst, wie sie in Listing 2.10 auf Seite 66 definiert ist, beschränken sich im Wesentlichen darauf, die Datenfelder des Template entsprechend zu füllen und Benutzeraktionen an den zentralen Service weiterzureichen. Eine Besonderheit, vor allem gegenüber der Komponente `AdminTranslationComponent`, liegt darin, dass diese Komponente bei einer Veränderung nicht automatisch neu erzeugt wird, da nicht wie bei `AdminTranslationComponent` eine select-Liste zurückgesetzt wird, sondern die Anzahl benötigter Instanzen meist trotz Änderung konstant ist. Demnach kommen der Methode `ngOnChanges` viele Initialisierungsaufgaben zu, während sie gleichzeitig auch manuell ausgelöst werden kann, um eine Aktualisierung zu erzwingen - zum Beispiel dann, wenn der Benutzer die gemachten Änderungen in den Formularfeldern zurücksetzen möchte. Ein Löschen der Sprache wird durch die Komponente so lange verhindert, bis es keine Übersetzungen mehr in dieser Sprache gibt. Das hilft dabei, verwaiste Datenreste nach dem Löschen zu vermeiden. Die Auswählbarkeit der Sprache umzuschalten ist hingegen jederzeit möglich.

Abbildung 3.3: Detailansicht der `AdminLanguageComponent` mit ihren Einstellmöglichkeiten. Eine Nachricht für nicht-verfügbare Übersetzungen wird nur benötigt, wenn die Sprache auch direkt auswählbar ist. Ein Button zum Speichern der Änderungen erscheint automatisch, sobald ein Wert verändert wird, während zum endgültigen Löschen ein zweiter Button betätigt werden muss, der erst nach Klick auf den ersten Button erscheint.

3.6 Übersetzungen verwalten mit AdminTranslationComponent

Auch die `AdminTranslationComponent` fasst viele Initialisierungsaufgaben in der Methode `ngOnChanges` (siehe Listing 2.12 auf Seite 70, Zeilen 35ff.) zusammen, wird aber deutlich öfter auch komplett neu instanziiert. Grundlegende Mechanismen funktionieren ähnlich wie bei der benachbarten Komponente `AdminLanguageComponent`, sodass auch das Template in Listing 2.13 (Seite 77) dem der anderen Komponente durchaus ähnelt.

Ein großer Unterschied besteht allerdings in der Anzahl möglicher Zustände und den zentralen Eingabefeldern für die übersetzten Daten. Hier sticht bei der Betrachtung von Abbildung 3.4 vor allem der Bezeichner vor dem standardsprachlichen Inhalt, oberhalb der Textfelder, ins Auge. An dieser Stelle findet sich der DOM-Pfad, an dem sich ablesen lässt, wie tief und vor allem in welche Elemente der Textinhalt ursprünglich verschachtelt war und nach der Übersetzung entsprechend wieder sein wird. Den DOM-Pfad berechnet die Komponente aus der DOM-Signatur in einem recht interessanten Verfahren, das in Listing 2.12 ab Zeile 99 nachvollzogen werden kann. Diese Maßnahme gehört untrennbar zur Kontextbeschaffung für Übersetzungen dazu und erfüllt damit die funktionale Anforderung **FA301** (Seite 10). Nach nochmaliger Bestätigung können einzelne Übersetzungsdaten gelöscht werden, woraufhin stattdessen eine leere Instanz zur Neuanlage erscheint.

Die Option „Übersetzung in alternative Sprache bevorzugen“ stellt eine elegante Möglichkeit dar, dem System mitzuteilen, dass eine Übersetzung zwar fehlt, aber nicht angefertigt werden muss. Da durch Gebrauch dieser Option sogar ohne Hinweis für den Benutzer auf die Standardsprache zurückgewechselt werden kann, ist diese Option zudem im Moment noch die einzige Möglichkeit, dynamische Daten in die Übersetzung zu transportieren ohne diese durch statische Übersetzungen zu überschreiben.

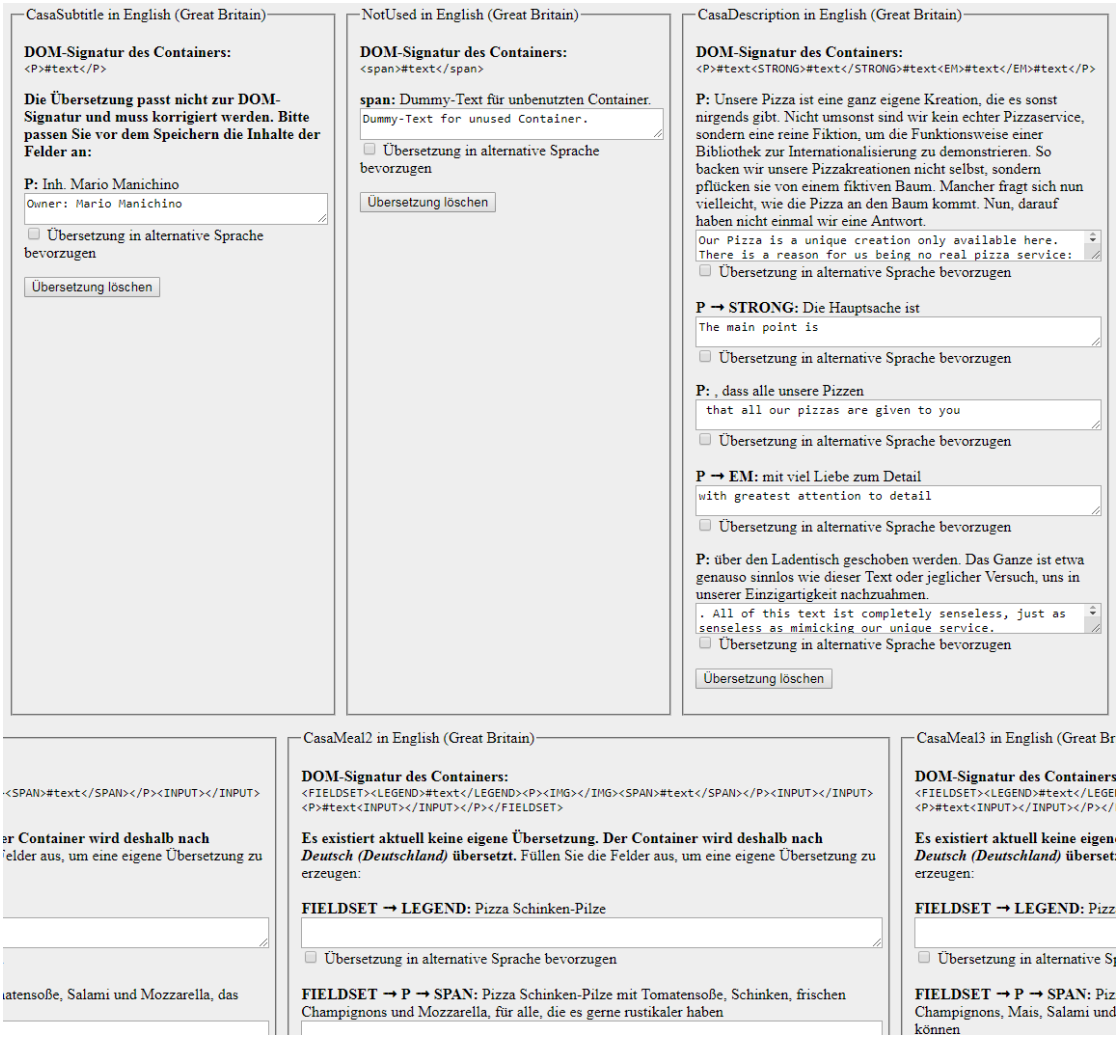

Abbildung 3.4: Die Detailansicht der `AdminTranslationComponent` zeigt große Ähnlichkeiten zur `AdminLanguageComponent` in punkto Bedienung. Eine große Besonderheit ist der aufgeschlüsselte DOM-Pfad an jedem bearbeitbaren Textknoten. Das Hinzufügen neuer Übersetzungen ist jederzeit möglich, da auch für nicht vorhandene Übersetzungen Instanzen angezeigt werden. Besonderes Augenmerk muss auch auf den Container links oben gerichtet werden. Hier stimmen die DOM-Signatur des Containers und der Übersetzung nicht mehr überein. Das kann passieren, wenn das DOM im Quellcode geändert wurde. Statt die Übersetzungsdaten zu verwerfen markiert das System den Container zur manuellen Überprüfung, liefert die alten, übersetzten Daten gleich mit und ordnet diese nach bestem Gewissen in der unpassenden Anzahl an Feldern an, damit sie für eine manuelle Korrektur herangezogen werden können.

3.7 Textdaten identifizieren und ersetzen

In den Abschnitten 2.3 und 2.4 wurden bereits viele Schlüsselkonzepte erläutert, die die Funktionalität der Klassen `TextContainer` und `TextualContent` sowie der Direktive `idDirective` betreffen. Dabei ist die in Listing 2.14 definierte Direktive schnell zu erfassen, da sie nur als Eintrittspunkt für die Klasse `TextContainer`, von der in Zeile 15 eine neue Instanz erzeugt wird, fungiert.

Bereits interessanter ist da der Vergleich der Vorüberlegungen aus dem vergangenen Kapitel mit der realen Umsetzung in den Klassen `TextContainer` und `TextualContent`. Abbildung 3.5 zeigt daher zunächst einen Vergleich zwischen den Übersetzungsergebnissen je nach eingestellter Sprache.

Vergleicht man die beiden englischsprachigen Darstellungen auf der rechten Seite, so fällt auf, dass hier die selben Übersetzungen vorhanden zu sein scheinen. Tatsächlich ist das allerdings nicht der Fall, denn eigentlich existiert für `en-US` keine einzige Übersetzung, wie der Blick in die Verwaltung zeigt. Das Bild, das dort beobachtet werden kann, ist ausschließlich ein Ausdruck davon, dass die im Abschnitt 2.4 angestellten Vorüberlegungen umgesetzt wurden, so dass sich die Übersetzung in `en-US` der Übersetzungsdaten von `en-GB` bedient, da beide Sprach-Tags in den ersten beiden Buchstaben übereinstimmen. Diese Form der Suche einer nach einer Alternativsprache ist, unter Rücksicht auf die Option `preferAltLang`, in der Definition von `TextualContent` in Listing 2.16 zu finden (Methode `switchLanguage`, Zeile 42ff). Der Algorithmus entspricht an der Stelle im Wesentlichen dem in Abschnitt 2.4 gegebenen Überblick.

Die Ersetzung der Textblöcke findet in der selben Methode statt, denn die Instanz von `TextualContent` bekommt über ihren Container die Übersetzungsdaten ausgeliefert und speichert diese dann in Form von *TextNodes* im Array `translatedTextNodes` zwischen. Wird nun eine Sprache gewählt, deren Übersetzungen schon dort zwischengespeichert sind, so wird ausschließlich der aktuell im DOM eingehängte Knoten `currentTextNode` von seinem übergeordneten Element gelöst bzw. durch den neuen Knoten ersetzt – das kann in Zeile 48ff. nachvollzogen werden. Sollte die Übersetzung nicht bereits bekannt sein, so wird sie neu angefordert.

Sollte es in diesem Prozess passieren, dass weder eine Übersetzung in der gewünschten Sprache noch eine alternative Übersetzung gefunden werden, obwohl kein Wunsch zum Vorzug alternativer Übersetzungen vorliegt, so wird das Problem an den `TextContainer` gemeldet und dessen Methode `switchToAlternativeLanguage` (Listing 2.15, Seite 80, Zeile 177ff.) aufgerufen. Diese ändert dann die Sprache in allen zu diesem Container gehörenden Textknoten zurück zur Standardsprache, um sprachliche Brüche mitten im Satz zu vermeiden, und platziert den bei der Sprachdefinition festgelegten Hinweis für fehlende Übersetzungspassagen in roter Schrift ans Ende des solchermaßen behandelten Containers. Diese Maßnahme musste in Abbildung 3.5 für die Sprache `fr-FR` offensichtlich oft ergriffen werden - denn bis auf einen Baustein ist noch kein Baustein in die französische Sprache übersetzt.

Doch der `TextContainer` hat noch mehr wichtige Aufgaben: Bei seiner Initialisierung analysiert er durch rekursiven Aufruf seiner Methode `exploreDOM` (Zeile 211ff.) die Struktur des `nativeElement` (in dem Fall also des Elements, das die Direktive *intmanId* trägt), erzeugt dabei auf jeder Verschachtelungsebene einen neuen `TextualContent` und berechnet dabei die `domSignature`, die für das automatisierte Erkennen von Veränderungen auf Quelltextebene und die daraus resultierende Invalidierung von Übersetzungsdaten unverzichtbar ist. Nach Berechnung der *domSignature* wird in Zeile 40ff. zunächst geprüft, ob auf dem Server aktuelle, auf diesen Container passenden Konfigurationsdaten vorliegen; falls nicht, werden die vom TextContainer bei seiner Initialisierung berechneten Werte als neue Konfiguration auf dem Server gespeichert. Dieser Mechanismus erfüllt die funktionalen Anforderungen **FA204** (Seite 9) und **FA205** (Seite 9).

Nach der Erzeugung der Instanzen von `TextualContent` werten diese den Inhalt des ihnen zur Verwaltung übergebenen *TextNode* aus und melden ihn als standardsprachliche Übersetzung an den `TextContainer` zurück. Dieser sammelt diese Meldungen zunächst mithilfe seiner Methode `registerDefaultTranslation` (Zeile 72ff.) bis alle Textknoten ihre Daten gemeldet haben. Dann vollzieht er einen Vergleich der gemeldeten Texte mit den am Server gespeicherten Übersetzungen in Defaultsprache und sorgt auch hier gegebenenfalls dafür, die standardsprachlichen Texte auf dem Server zu aktualisieren.

Da die Übersetzungen auf dem Server nach *TextContainer* und Sprache gebündelt gespeichert sind und ein Sprachwechsel für gewöhnlich je Container einheitlich passiert, wird auch die Abfrage und Zwischenspeicherung der übersetzten Texte durch den *TextContainer* übernommen.

Abbildung 3.5: Die Testumgebung *Casa del Diavolo* in verschiedenen Sprachen. Angefangen mit *Deutsch (Deutschland)* (`de-DE`, oben links) sind dort noch die übersetzten Seiten in *English (Great Britain)* (`en-GB`, oben rechts), *English (United States)* (`en-US`, oben links) und *Français (France)* (`fr-FR`, unten links) zu sehen. Die roten Textanteile kennzeichnen Passagen, bei denen aufgrund fehlender Übersetzungen in die Standardsprache gewechselt werden musste.

3.8 Die zentrale Datenverwaltung

Obwohl der zentrale Service `IntManLibService` an fast allen Einzelteilen der Bibliothek direkt angebunden ist und der Umfang des Quelltextes, der zu seiner Realisierung nötig ist und in Abschnitt A.2.10 (Seite 94) teilweise eingesehen werden kann, alle anderen Teile der Bibliothek bei weitem übertrifft, ist er vermutlich, was die Analyse der Implementierung angeht, eher wenig ergiebig. Das liegt daran, dass in diesem Service vor allem Helferfunktionalitäten gebündelt sind. So besitzt der Service einige Arrays, in denen er die Einzelteile der Bibliothek während ihres Lifecycle registriert hält. Für jedes dieser Arrays ist je eine Methode vorgesehen, die die Eintragung oder Austragung eines Objekts übernimmt. Weiterhin gibt es einen *getter* und *setter* für die aktuelle Sprache `curLang`.

Ein wenig interessanter sind da noch die Funktionen, die HTTP-Requests in Richtung der REST-API durchführen. So sind für alle Datenklassen (vergleiche Abschnitt A.2.2 auf Seite 55 verschiedene **GET**ter (typischerweise um einzelne oder alle Daten dieses Typs abzufragen) vorhanden, aber auch weitreichendere HTTP-Methoden wie **POST**, **PUT** und **DELETE** werden dort zur Verfügung gestellt. Bei den Letztgenannten wird grundsätzlich nach erfolgreichem Abschluss des Request die interne Datenbasis aktualisiert, entweder durch direktes Anwenden der Veränderung oder durch eine Aufforderung an alle registrierten Bibliotheksteile, eine Aktualisierung ihrer Daten und eine Löschung ihrer Caches durchzuführen. Diese sehr individuellen Operationen auf den Arrays mit registrierten Elementen gehören zu den anspruchsvollsten Teilen der Implementierung dieses Services; sie sind beispielhaft ab Zeile 103 im Listing 2.17 zu finden.

Die letzte wichtige Aufgabe kommt dem `IntManLibService` in der Form zu, dass er beim Sprachwechsel via `SwitcherComponent` die neu ausgewählte Sprache entgegen nimmt und die Ersetzung der Texte in den bei ihm registrierten Containern anstößt.

4

Ausblick und Schlussbemerkungen

Im Verlauf dieser Arbeit wurde ein Prototyp vorgestellt, analysiert und dokumentiert, der einen Beitrag zur Lösung der in Abschnitt 1.1 genannten Problemstellung liefern soll.

Es ist klar, dass das Problem an sich alleine durch diesen Prototypen nicht gelöst wird. So bleiben auch aus den in Abschnitt 2.1 bereits genannten Anforderungen trotz der vieler bereits implementierter Funktionalitäten einige übrig, die zur Zeit durch den Prototypen noch nicht umgesetzt werden. Im Folgenden sollen diese bisher scheinbar unbeachteten Anforderungen noch einmal in den Fokus gerückt werden.

4.1 Bislang nicht erfüllte Anforderungen

Die funktionale Anforderung **FA105** (Seite 8) fordert, dass automatisch formatierte Angaben gemäß der aktuell ausgewählten Sprache ausgegeben werden. Tatsächlich ist der Prototyp im Moment insgesamt noch sehr schlecht auf automatisch formatierte oder auch dynamisch veränderliche Texte eingestellt. Es ist zwar bislang prinzipiell schon möglich, auch trotz solcher Inhalte eine Übersetzung eines Containers anzufertigen, man muss dafür aber dann jeweils aktiv in allen Sprachen angeben, dass dieser Textknoten in einer Alternativsprache ausgeliefert werden soll - denn erst dann bleibt er alleine in der Standardsprache und kann dem Anspruch, den man an ein *interpolation binding* oder ähnliches richtet, auch gerecht werden.

Wünschenswert wäre es, eine weitere Direktive oder ein entsprechendes Attribut einzurichten, das die Übersetzung für einen einzelnen Textknoten außer Kraft setzt. Dann könnte der Textknoten zum Erhalt des Kontext weiter bekannt sein und in der Admi-

nistrationsansicht auch eingesehen werden, ansonsten aber von jeglicher Ersetzung befreit werden. Gerade bei dynamisch veränderbaren Daten ist diese Ausnahme sehr wichtig, da sich ansonsten die unnötigen Server-Anfragen häufen, da veränderte standardsprachliche Texte, also auch dynamische Inhalte, ja zum Zwecke der Aktualisierung der Datensätze regelmäßig an den Server gemeldet werden.

Anforderung **FA202** (Seite 9) wurde nicht wie ursprünglich angefordert implementiert; diese Funktionalität wird statt der geforderten manuellen Verwaltung nun allerdings durch einen automatischen Vorgang (vgl. **FA204**) versehen, so dass der Kern der Funktionalität bereits implementiert ist. Das selbe trifft auch auf Anforderung **FA205** zu, die zwar im Kern implementiert ist, aber keine manuelle Bedienung ermöglicht, sondern ausschließlich automatisiert abläuft.

Die Anforderung **FA207** zum Machen intelligenter Vorschläge wurde als Komfortfunktion identifiziert. Hier gibt es bisher nur ein Konzept, aber keine Implementierung. Das Konzept sieht vor, dass dann, wenn für einen Textknoten keine Übersetzung vorliegt, zusätzlich zum leeren Eingabefeld Buttons mit Vorschlägen in der *AdminTranslationComponent* erscheinen. Diese Vorschläge setzen sich zusammen aus anderen Übersetzungen, die zufällig die selbe oder eine ähnliche standardsprachliche Übersetzung haben sowie aus Vorschlägen durch extern angebundene Services.

Ähnliches trifft auch auf **FA210**, den Excel-Export und Import durch. Es gibt für Angular bereits Bibliotheken, die Dateiexporte ermöglichen. Eine solche müsste eingebunden werden und mit zu *AdminTranslationComponent* analogen Funktionalitäten ausgestattet werden. Da der Prototyp seine Funktion als *proof of concept* auch mit Basisfunktionalitäten bereits erfüllt, wurde aus Effizienzgründen auf die Implementierung von Komfortfunktionen teilweise bewusst verzichtet. Hinzu kommt, dass es neben Excel-Dateien sinnvollere und gerade unter Übersetzern weiter verbreitete Formate gibt, die dann sinnvoller Weise auch unterstützt werden sollten.

Hinsichtlich der nicht-funktionalen Anforderung **NF101** wurde bereits vieles richtig gemacht, es kommt aber in diesem Themenkomplex auch ein Versäumnis vor, das eigentlich als eigene Anforderung in die Anforderungsanalyse einfließen hätte müssen. Der Prototyp ersetzt momentan zwar Inhalt, macht die verwendete Sprache aber nicht

in den dazugehörigen oder übergeordneten Tags durch eine Kennzeichnung mit dem HTML-Attribut `lang` deutlich. Das offenbart eine Lücke, die der Prototyp im Moment noch vorweist: Der Prototyp ist im Moment nicht barrierefrei und macht veränderte Sprachen nicht sauber kenntlich, was aber zu seinen Grundaufgaben gehören würde. Es genügt hier allerdings nicht, einfach nur `lang`-Attribute automatisiert zu setzen, denn je nach Container und individuellem Status der Textknoten kann das angemessene `lang`-Attribut durchaus von der Sprache abweichen. Besonders deutlich wird das bei Fremdwörtern. So wäre beispielsweise in einem deutschen Text über Fast-Food das Wort Burger sinnvollerweise mit *lang=en* zu kennzeichnen, obwohl es sich um den Text in deutscher Sprache handelt. Hier muss also noch ein sinnvolles Konzept zum Umgang mit *lang*-Angaben gefunden werden.

4.2 Schlusswort

Trotz einiger Lücken, die der Prototyp noch aufweist, sind die Konzeption und die Grundidee als Erfolg und als erster Schritt in die richtige Richtung zu werten. So ist der überwiegende Anteil der formulierten Anforderungen bereits zufriedenstellend implementiert und für die nicht implementierten Funktionalitäten gilt, dass es sich bei ihnen nicht um Basisfunktionalitäten handelt; zudem ist bei keiner dieser Anforderungen eine konzeptionelle Unverträglichkeit mit dem aktuellen Prototypen gegeben.

Ein gewisses Interesse weckt auch die Frage, inwiefern der Prototyp in seinem momentanen Umfang bereits produktiv nutzbar wäre – dem wollen wir aber zumindest an dieser Stelle nicht auf den Grund gehen.

In der Hoffnung, durch den angefertigten Prototypen und diese Dokumentation etwas sinnvolles, wenn auch vielleicht teilweise unkonventionelles zur Lösung eines weit verbreiteten Problems beigetragen zu haben, schließen wir, wie wir begonnen haben – mit einem Zitat von Ludwig Wittgenstein:

> „*Die Idee sitzt gleichsam als Brille auf unsrer Nase, und was wir ansehen, sehen wir durch sie. Wir kommen gar nicht auf den Gedanken, sie abzunehmen.*“

Literaturverzeichnis

[1] Robertson, S., Robertson, J.: Mastering the requirements process: getting requirements right. 3. ed. edn. Addison-Wesley, Upper Saddle River, NJ ; Munich [u.a.] (2013) Bibliotheks-Zentrale.

[2] Foster, E.C.: Software engineering: a methodical approach. Online-ausg. edn. Apress, Berkeley, CA (2014)

[3] Angular Contributors, Google LLC., C.B..: Angular Tour of Heroes: Tutorial. https://angular.io/tutorial (2018) abgerufen am 17.10.2018.

[4] Aden, D.: Visually Representing Angular Applications - Mozilla Hacks Blog. https://hacks.mozilla.org/2014/11/visually-representing-angular-applications/ (2014) abgerufen am 13.08.2018.

[5] A. Phillips, M. Davis, I.N.W.G.: Tags for Indentifying Languages (BCP47). https://tools.ietf.org/html/bcp47 (2009) abgerufen am 20.10.2018.

[6] Alvestrand, H.: Tags for the Identification of Languages (RFC1766). https://tools.ietf.org/html/rfc1766 (1995) abgerufen am 20.10.2018.

[7] Palmer, T.: Angular In Depth: The Angular Library Series. https://blog.angularindepth.com/creating-a-library-in-angular-6-87799552e7e5 (2018) abgerufen am 19.08.2018.

[8] Larsen, H.: Dokumentation zu Angular CLI: Library support in Angular CLI 6. https://github.com/angular/angular-cli/wiki/stories-create-library (2018) abgerufen am 02.09.2018.

[9] Tuzi, C.: Medium: How to build and publish an Angular module. https://medium.com/@cyrilletuzi/how-to-build-and-publish-an-angular-module-7ad19c0b4464 (2016) abgerufen am 02.09.2018.

[10] Chenkie, R.: Angular Authentication: Using the Http Client and Http Interceptors. https://ryanchenkie.com/angular-authentication-using-the-http-client-and-http-interceptors (2017) abgerufen am 21.10.2018.

[11] Richard Ishida, J.Y.: Using select to link to localized content. https://www.w3.org/International/questions/qa-navigation-select.en (2014) abgerufen am 21.10.2018.

[12] Bittersmann, G.: Über Sprachen und Flaggen. https://bittersmann.de/articles/no-flags/ (2011) abgerufen am 21.10.2018.

A

Quelltexte

Im Folgenden sollen die Quelltexte der Bibliothek und ihrer Testumgebung aufgelistet werden. Dabei ist jeder Quelltext mit einer kurzen Erläuterung versehen. Da in einigen Fällen leicht modifizierte Versionen der ursprünglichen Quelltexte hier abgedruckt sind, sei zusätzlich auf das *github*-Repository mit den tagesaktuellen, vollständigen Quelltexten hingewiesen: `https://github.com/campingrider/ng-int-man-dummy`

Der Stand der Bearbeitung zum Zeitpunkt der Abgabe dieser Arbeit entspricht dem Release **v0.5.0-beta**, der unter der URL `https://github.com/campingrider/ng-int-man-dummy/tree/v0.5.0-beta` im Detail abgerufen werden kann.

A.1 Basisversion – Testumgebung Casa del Diavolo

Listing 1.1: Quelltext der Datei `casabasis.html`

```
<main style="text-align:center">
  <header>
    <h1>
      Willkommen bei {{ title }}!
    </h1>
    <p>{{subtitle}}</p>
  </header>
```

```
8    <p>Unsere Pizza ist eine ganz eigene Kreation, die [...] eine
     ↪  Antwort. <strong>Die Hauptsache ist</strong>, dass alle
     ↪  unsere Pizzen <em>mit viel Liebe zum Detail</em> über den
     ↪  Ladentisch [...] Einzigartigkeit nachzuahmen.</p>
9    <h1>Online-Bestellung</h1>
10   <p>Wählen Sie aus unserem reichhaltigen Angebot.</p>
11   <form method="GET" class="order">
12     <div class="meals">
13       <fieldset class="meal" *ngFor="let meal of meals; let
         ↪  m=index">
14         <legend>{{meal.name}}</legend>
15         <p><img src="{{meal.icon}}"/><span>{{
           ↪  meal.description}}</span></p>
16         <input type="hidden" [name]="'names['+m+']'"
           ↪  value="{{meal.name}}"/>
17         <p *ngIf="meal.extras.length>0">Wählen Sie zusätzlich aus
           ↪  folgenden Extras:</p>
18         <ul *ngIf="meal.extras.length>0">
19           <li *ngFor="let extra of meal.extras">
20             <input type="checkbox" [name]="'extras['+m+']'"
               ↪  value="{{extra}}"/><span>{{extra}}</span>
21           </li>
22         </ul>
23         <p>Gewünschte Anzahl: <input type="number"
           ↪  [name]="'quantity['+m+']'" value="0"/></p>
24       </fieldset>
25     </div>
26     <fieldset class="buttons">
27       <button>Bestellung aufgeben →</button>
28     </fieldset>
29   </form>
30 </main>
```

Basisversion des HTML-Templates für Casa del Diavolo, bislang noch ohne bibliotheksspezifische Änderungen. Dargestellt ist das *main*-Element, das im oberen

Bereich den *header* mit dem Seitentitel enthält und darunter den Bestellbereich, der in ein *form*-Element eingebettet ist. Im *form*-Element wird durch Angular dynamisch je bestellbarem Gericht ein *fieldset*-Element bereitgestellt, das die verschiedenen Eigenschaften des Gerichts in Formularelementen wiedergibt.

A.2 Die Quelltexte der Bibliothek

A.2.1 Modul und Public-API der Bibliothek

Listing 2.1: Quelltext der Datei `int-man-lib.module.ts`

```
1  import { NgModule } from '@angular/core';
2  import { AdminComponent } from './admin/admin.component';
3  import { IdDirective } from './id.directive';
4  import { SwitcherComponent } from './switcher/switcher.component';
5  import { CommonModule } from '@angular/common';
6  import { FormsModule } from '@angular/forms';
7  import { HttpClientModule } from '@angular/common/http';
8  import { AdminLanguageComponent } from
   ↪  './admin-language/admin-language.component';
9  import { AdminTranslationComponent } from
   ↪  './admin-translation/admin-translation.component';
10
11 @NgModule({
12   imports: [
13     CommonModule,
14     FormsModule,
15     HttpClientModule
16   ],
17   declarations: [
18     AdminComponent,
19     IdDirective,
20     SwitcherComponent,
```

```
21      AdminLanguageComponent,
22      AdminTranslationComponent],
23    exports: [AdminComponent, SwitcherComponent, IdDirective]
24  })
25  export class IntManLibModule { }
```

Das *ngModule* der Bibliothek *int-man-lib* deklariert die zugehörigen Komponenten und Services. Darunter besonders ausgezeichnet sind die Komponenten `AdminComponent` und `SwitcherComponent` sowie die Direktive `IdDirective` (*IntmanId*), die zudem noch durch das Modul exportiert werden.

Listing 2.2: Quelltext der Datei `public-api.ts`

```
1  /*
2   * Public API Surface of int-man-lib
3   */
4
5  export * from './lib/int-man-lib.service';
6  export * from './lib/int-man-lib.module';
7
8  export * from './lib/admin/admin.component';
9  export * from './lib/switcher/switcher.component';
10 export * from './lib/id.directive';
```

Die *public_api* der Bibliothek *int-man-lib* kennzeichnet die Eigenschaften der exportierten Komponenten `AdminComponent` und `SwitcherComponent` sowie der Direktive `idDirective` (*intmanId*), damit die einbindende Anwendung darauf Zugriff erhält.

A.2.2 Klassen zur Repräsentation von Daten

Listing 2.3: Quelltext der Datei `container-setting.ts`

```
1 export class ContainerSetting {
2   id: string;
3   domSignature: string;
4   contains: number;
5 }
```

Die Datenklasse `ContainerSetting` repräsentiert die Rahmendaten eines *Text-Container*. Sie enthält die `id` des Container, seine zuletzt bekannte `domSignature` und die Anzahl der enthaltenen Textknoten (`contains`). Es handelt sich um eine Klasse, die so auch in der Datenbank bzw. an der REST-Schnittstelle vorkommt und deshalb eine wichtige Rolle als Austauschformat spielt.

Listing 2.4: Quelltext der Datei `language.ts`

```
1 export class Language {
2   public id: string;
3   public title: string;
4   public unavailableText: string;
5   public selectable: boolean;
6
7   constructor(id: string, title: string = id) { this.id = id;
    ↪  this.title = title; }
8 }
```

Die Datenklasse `Language` repräsentiert eine Sprache. Sie enthält die `id` der Sprache nach RFC1766, ihren Anzeigetitel (z.B. für die Administrationsansicht oder die Sprachauswahl), einen Text mit Bezeichnung `unavailableText`, der an jeder Textpassage angezeigt werden soll, die nur in einer Alternativsprache angezeigt werden kann und eine boolesche Variable `selectable`, die angibt, ob die Sprache zur Sprachauswahl angezeigt werden soll. Es handelt sich um eine Klasse, die so auch in der Datenbank bzw. an der REST-Schnittstelle vorkommt und deshalb eine wichtige Rolle als Austauschformat spielt.

Listing 2.5: Quelltext der Datei `translation.ts`

```
export class Translation {
  public id: string;
  public containerId: string;
  public contents: string[];
  public langId: string;
  public preferAltLang: boolean[];
}
```

Die Datenklasse `Translation` repräsentiert die Übersetzungsdaten für die Inhalte eines *TextContainer* mit Bezeichner `containerId` in die Sprache mit Sprachtag `langId`. Die übersetzten Inhalte der einzelnen Textknoten liegen in der Eigenschaft `contents` vor, wobei für jeden der Textknoten im Array `preferAltLang` gespeichert ist, ob das System stattdessen eine alternativsprachlichen Übersetzung bevorzugen soll. Die `id` einer Translation setzt sich aus der *containerId* und der *langId* zusammen, verbunden durch einen Bindestrich.

A.2.3 Die Komponente SwitcherComponent

Listing 2.6: Quelltext der Datei `switcher.component.ts`

```
import { IntManLibService } from './../int-man-lib.service';
import { Component, OnInit, Input, HostListener, OnDestroy } from
↪  '@angular/core';
import { Language } from '../language';

@Component({
  selector: 'intman-switcher',
  templateUrl: './switcher.component.html',
  styleUrls: ['./switcher.component.css']
})
export class SwitcherComponent implements OnInit, OnDestroy {

```

```
12    langs: Language[];
13
14    @Input() variant = 'dropdown';  // possible variants: dropdown,
      ↪  select, radio, buttons
15    @Input() lang: string;
16
17    constructor(private intManLibService: IntManLibService) { }
18
19    ngOnInit() {
20      this.intManLibService.getLanguages().subscribe(languages =>
        ↪  this.langs = languages.filter(lang => lang.selectable));
21      this.intManLibService.defLang.subscribe(lang => this.lang =
        ↪  lang.id);
22      this.intManLibService.registerSwitcherComponent(this);
23    }
24    ngOnDestroy() {
25      this.intManLibService.unregisterSwitcherComponents(this);
26    }
27    @HostListener('change', ['$event.target'])
28    onClick(target) {
29      this.intManLibService.getLanguage(this.lang).subscribe(lang =>
        ↪  this.intManLibService.setLanguage(lang));
30    }
31    buttonClick(lang) {
32      this.intManLibService.getLanguage(lang).subscribe(l =>
        ↪  this.intManLibService.setLanguage(l));
33    }
34  }
```

Die Komponente `SwitcherComponent` wird durch die einbindende Anwendung überall dort eingebunden, wo ein GUI-Element zur Sprachwahl erwünscht ist. Bei der Einbindung kann über das Attribut `variant` ein bestimmter Stil vorgegeben werden, darunter *dropdown*, *select*, *radio* und *buttons*. Die aktuell ausgewählte Sprache wird unter `lang` gespeichert. Die Komponente wird bei der Initialisierung im zentralen Service registriert und bei der Destruktion dort auch wieder abgemeldet. Anfangs

wird außerdem die Standardsprache abgefragt und entsprechend eingestellt. Die Komponente reagiert auf einen Usereingaben und reicht die gewählte Sprache an den zentralen Service weiter.

Listing 2.7: Quelltext der Datei `switcher.component.html`

```
1  <aside *ngIf="variant==='dropdown'">
2    <p intmanId="IntManInternalChooseLanguage">Bitte wählen Sie Ihre
     ↪  Sprache:</p>
3    <select size='1' [(ngModel)]="lang">
4      <option *ngFor="let l of langs"
       ↪  value="{{l.id}}">{{l.title}}</option>
5    </select>
6  </aside>
7
8  <aside *ngIf="variant==='radio'">
9    <fieldset>
10     <legend intmanId="IntManInternalChooseLanguage">Bitte wählen
       ↪  Sie Ihre Sprache:</legend>
11     <p *ngFor="let l of langs; let i=index">
12       <input type="radio" name="IntManSwitcherRadio"
         ↪  value="{{l.id}}" id="IntManSwitcherRadio{{i}}"
         ↪  [(ngModel)]="lang">
13       <label for="IntManSwitcherRadio{{i}}">{{l.title}}</label>
14     </p>
15   </fieldset>
16 </aside>
17
18 <aside *ngIf="variant==='select'">
19   <p intmanId="IntManInternalChooseLanguage">Bitte wählen Sie Ihre
     ↪  Sprache:</p>
20   <select size='{{langs.length}}' [(ngModel)]="lang">
21     <option *ngFor="let l of langs"
       ↪  value="{{l.id}}">{{l.title}}</option>
22   </select>
```

```
</aside>

<aside *ngIf="variant==='buttons'">
  <p intmanId="IntManInternalChooseLanguage">Bitte wählen Sie Ihre Sprache:
    <button *ngFor="let l of langs" (click)="buttonClick(l.id)">{{l.title}}</button>
  </p>
</aside>
```

Das Template der Komponente `SwitcherComponent` ist vor allem aufgrund des je nach Einstellung von *variant* unterschiedlichen Markups interessant. In allen vier Varianten wird über die Zahl der Sprachen iteriert, die dabei erzeugten Elemente unterscheiden sich allerdings - von Dropdown-Optionen über Buttons und Mehrfachauswahl.

A.2.4 Die Komponente AdminComponent

Listing 2.8: Quelltext der Datei `admin.component.ts`

```
import { Component, OnInit, OnDestroy, Input } from '@angular/core';
import { IntManLibService } from '../int-man-lib.service';
import { Language } from '../language';
import { TextContainer } from '../text-container';
import { ContainerSetting } from '../container-setting';

@Component({
  selector: 'intman-admin',
  templateUrl: './admin.component.html',
  styleUrls: ['./admin.component.css']
})
export class AdminComponent implements OnInit, OnDestroy {

```

```
14    public messages: string[];
15    public langs: Language[];
16    public containerSettings: ContainerSetting[];
17    public missingTranslations: Object[];
18    public changedSomethingOnNewLang = false;
19    public newLangMeetsRequirements = false;
20    public newLangId = '';
21
22    @Input() langId = '';
23    @Input() containerId = '';
24
25    constructor(private intManLibService: IntManLibService) { }
26
27    ngOnInit() {
28      this.intManLibService.registerAdminComponent(this);
29      this.intManLibService.getLanguages().subscribe(langs =>
        ↪  this.langs = langs);
30      this.intManLibService.getAllContainerSettings().subscribe(
        ↪  containerSettings => this.containerSettings =
        ↪  containerSettings);
31      this.collectMissingTranslations();
32    }
33    ngOnDestroy() {
34      this.intManLibService.unregisterAdminComponent(this);
35      this.changedSomethingOnNewLang = false;
36    }
37    /** collects missing translations */
38    public collectMissingTranslations() {
39      this.missingTranslations = [];
40      this.intManLibService.getAllContainerSettings().subscribe(
        ↪  containers =>
        ↪  {
41        this.intManLibService.getLanguages().subscribe(langs => {
42          langs = langs.filter(l => l.selectable);
43          const langIds = langs.map(l => l.id);
```

```
44          containers.forEach(container => {
45            this.intManLibService.getAllTranslations(container.id).subscribe(translations => {
46              this.missingTranslations = this.missingTranslations.filter(mT => mT['containerId'] !== container.id);
47              translations = translations.filter(
48                translation => translation.contents.length === container.contains &&
49                langIds.indexOf(translation.langId) >= 0
50              );
51              const translationLangIds = translations.map(t => t.langId);
52              if (translations.length < langIds.length) {
53                const missing = [];
54                langIds.forEach(lid => { if (translationLangIds.indexOf(lid) < 0) { missing.push(lid); }});
55                this.missingTranslations.push( {
56                  'containerId': container.id,
57                  'langTitles': langs.filter(l => missing.indexOf(l.id) >= 0).map(l => l.title)
58                } );
59              }
60            });
61          });
62        });
63      }
64      /** returns language object for given langId */
65      public findLang(langId: string) {
66        if (this.langs === undefined) { return undefined; }
67        return this.langs.find(tlang => tlang.id === langId);
68      }
```

```
69    /** returns containerSetting object for given containerId */
70    public findContainerSetting(containerId: string) {
71      if (this.containerSettings === undefined) { return undefined;
        ↪  }
72      return this.containerSettings.find(tSetting => tSetting.id ===
        ↪  containerId);
73    }
74    /** change status if there was some change */
75    public detectChangeOnNewLang(): boolean {
76      this.changedSomethingOnNewLang = true;
77      this.newLangMeetsRequirements = this.newLangId.substr(0, 2)
        ↪  === 'i-' || this.newLangId.substr(0, 2) === 'x-' ||
        ↪  this.newLangId.substr(2, 1) === '-';
78      return this.newLangMeetsRequirements;
79    }
80    /** add a new language */
81    public addLang(): void {
82      if (this.changedSomethingOnNewLang === true &&
        ↪  this.detectChangeOnNewLang()) {
83        const newLang = new Language(this.newLangId, 'Neue
          ↪  Sprache');
84        newLang.selectable = false;
85        this.intManLibService.addLanguage(newLang).subscribe(_ =>
          ↪  this.newLangId = '');
86      }
87    }
88  }
```

Die Komponente `AdminComponent` verfügt über Zwischenspeicher (`langs`, `messages`, `containerSettings`) für die wichtigsten Übersetzungsdaten für die Anwendung. Weiterhin werden von ihr die Werkzeuge zum Erstellen neuer Sprachen direkt bereitgestellt, außerdem gibt es veränderbare Speicher für die aktuell ausgewählte Sprache (`langId`) und den aktuell ausgewählten Container (`containerId`) für die gezielte Anzeige der untergeordneten Komponenten. Die Komponente ist während ihres Lifecycle beim zentralen Service registriert. Eine Besonderheit ist

die Methode `collectMissingTranslations`, die mit verhältnismäßig großem Aufwand alle bekannten TextContainer auf fehlende Übersetzungen prüft.

Listing 2.9: Quelltext der Datei `admin.component.html`

```
<section class="intmanAdmin">

  <h1>Internationalisation Manager: Verwaltung</h1>

  <details class="intmanAdminMessages intmanAdminTileContainer">
    <summary>
      <h2>Hinweise zu fehlenden Übersetzungen</h2>
    </summary>
    <form class="intmanAdminTileContainer">
      <div *ngFor="let m of missingTranslations"
       ↪  class="intman-tile">
        <fieldset>
          <legend>Container {{m.containerId}}</legend>
          <p>In folgenden auswählbaren Sprachen ist keine
           ↪  Übersetzung vorhanden:</p>
          <ul>
            <li *ngFor="let lT of m['langTitles']">{{lT}}</li>
          </ul>
        </fieldset>
      </div>
    </form>
  </details>

  <details>
    <summary>
      <h2>Sprachen: Verwaltung</h2>
    </summary>
    <form class="intmanAdminLanguages intmanAdminTileContainer">
      <intman-admin-language *ngFor="let l of langs"
       ↪  [lang]="l"></intman-admin-language>
```

```
28        <div class="intman-tile">
29          <fieldset>
30            <legend>Neue Sprache anlegen</legend>
31            <p><label>Sprach-ID nach <a
               ↪  href="https://tools.ietf.org/html/rfc1766">RFC1766</a>:
               ↪  <input type="text" name="newLangId"
               ↪  [(ngModel)]="newLangId"
               ↪  (keyup)="detectChangeOnNewLang()"
               ↪  (input)="detectChangeOnNewLang()"
               ↪  (change)="detectChangeOnNewLang()"></label></p>
32            <p>Die Sprach-ID kann im Nachhinein
               ↪  <strong>nicht</strong> geändert werden.</p>
33            <p class="intman-language-buttons
               ↪  intman-buttons"><button
               ↪  *ngIf="changedSomethingOnNewLang &&
               ↪  newLangMeetsRequirements"
               ↪  (click)="addLang()">Änderungen
               ↪  speichern!</button></p>
34            <p *ngIf="!newLangMeetsRequirements">Die ID muss den
               ↪  Regeln des RFC1766 entsprechen.</p>
35          </fieldset>
36        </div>
37      </form>
38    </details>
39
40
41    <details>
42      <summary>
43        <h2>Übersetzungen nach Textblock</h2>
44      </summary>
45      <p>Wählen Sie einen Textblock:
46        <select size="1" [(ngModel)]="containerId">
47          <option value="">-- Bitte wählen --</option>
48          <option *ngFor="let c of
             ↪  containerSettings">{{c.id}}</option>
```

```
49        </select>
50      </p>
51      <form class="intmanAdminTranslations intmanAdminTileContainer"
        ↪  *ngIf="findContainerSetting(containerId)!==undefined">
52        <intman-admin-translation *ngFor="let l of langs" [lang]="l"
          ↪  [containerSetting]="findContainerSetting(
          ↪  containerId)"></intman-admin-translation>
53      </form>
54    </details>
55
56    <details>
57      <summary>
58        <h2>Übersetzungen nach Sprache</h2>
59      </summary>
60      <p>Wählen Sie eine Sprache:
61        <select size="1" [(ngModel)]="langId">
62          <option value="">-- Bitte wählen --</option>
63          <option *ngFor="let l of langs"
            ↪  value="{{l.id}}">{{l.title}}</option>
64        </select>
65      </p>
66      <form class="intmanAdminTranslations intmanAdminTileContainer"
        ↪  *ngIf="findLang(langId)!==undefined">
67        <intman-admin-translation *ngFor="let c of
          ↪  containerSettings" [lang]="findLang(langId)"
          ↪  [containerSetting]="c"></intman-admin-translation>
68      </form>
69    </details>
70
71  </section>
```

Das Template der `AdminComponent` kennt vier Bereiche: Die Hinweise zu fehlenden Übersetzungen, die durch die Komponente selbst ermittelt werden, die Sprachverwaltung unter Einbeziehung der Unterkomponente `intman-admin-language` sowie eines Formularfeldes zur Erzeugung einer neuen Sprache und die beiden Be-

reiche *Übersetzungen nach Textblock* und *Übersetzungen nach Sprache*, in denen jeweils die Unterkomponente `intman-admin-translation` eingesetzt wird. Im Fall der beiden letzten Bereiche wird die Auswahl eines speziellen Textblocks bzw. die Auswahl einer speziellen Sprache durch *select*-Elemente realisiert, während über die jeweils anderen Werte mittels `*ngFor` iteriert wird.

A.2.5 Interne Komponente AdminLanguageComponent

Listing 2.10: Quelltext der Datei `admin-language.component.ts`

```
1  import { Component, OnInit, Input, OnDestroy, OnChanges } from
   ↪  '@angular/core';
2  import { Language } from '../language';
3  import { IntManLibService } from '../int-man-lib.service';
4
5  @Component({
6    selector: 'intman-admin-language',
7    templateUrl: './admin-language.component.html',
8    styleUrls: ['./admin-language.component.css']
9  })
10 export class AdminLanguageComponent implements OnInit, OnDestroy,
   ↪  OnChanges {
11
12   @Input() lang: Language;
13
14   private initialLang: Language;
15   public changedSomething = false;
16   public isDefLang = false;
17   public deletionRequested = false;
18   public deletionPossible = false;
19
20   constructor(private intManLibService: IntManLibService) { }
21
```

```
22    ngOnInit() {
23      this.intManLibService.registerLanguageComponent(this);
24    }
25    ngOnChanges() {
26      this.deletionPossible = false; this.deletionRequested = false;
27      if (this.initialLang === undefined) {
28        this.initialLang = new Language(this.lang.id,
          ↪ this.lang.title);
29        this.initialLang.selectable = this.lang.selectable;
30        this.initialLang.unavailableText =
          ↪ this.lang.unavailableText;
31      }
32      this.changedSomething = false;
33      this.intManLibService.defLang.subscribe(defLang =>
        ↪ this.isDefLang = defLang.id === this.lang.id);
34    }
35    ngOnDestroy() {
36      this.intManLibService.unregisterLanguageComponent(this);
37    }
38    /** request deletion - calculate whether deletion is possible */
39    public requestDeletion() {
40      this.deletionPossible = false;
41      this.deletionRequested = true;
42      this.intManLibService.getTranslationsByLanguage(
        ↪ this.lang.id).subscribe(
43        translations => { if (translations.length === 0) {
          ↪ this.deletionPossible = true; } }
44      );
45    }
46    /** delete language on user input */
47    public deleteLanguage() {
48      if (this.deletionRequested && this.deletionPossible) {
49        this.intManLibService.deleteLanguage(this.lang).subscribe(_
          ↪ => this.lang = undefined);
50      }
```

```
51    }
52    /** change status if there was some change */
53    public detectChange(): void { this.changedSomething = true; }
54    /** resets form data */
55    public reset(): void { this.lang = this.initialLang;
      ↪  this.initialLang = undefined; this.ngOnChanges(); }
56    /** saves current form data to server */
57    public save(): void {
58      // save object to server
59      this.intManLibService.updateLanguage(this.lang).subscribe();
60    }
61  }
```

Die Komponente `AdminLanguageComponent` nimmt Sprachdaten von der übergeordneten Komponente auf und stellt diese in einem Formular zur Bearbeitung dar. Insbesondere prüft sie, ob es sich um die Standardsprache handelt und legt diese Information in `isDefLang` ab. Die Komponente aktualisiert sich bei einer Veränderung selbsttätig und ist im zentralen Service registriert. Ein Speicher-Button wird angezeigt sobald eine Veränderung detektiert wurde (`changedSomething`), das Löschen ist auch nach nochmaliger Bestätigung (`deletionRequested` / `requestDeletion`) nur dann möglich, wenn es für die Sprache keine Übersetzungen mehr gibt (`deletionPossible`).

Listing 2.11: Quelltext der Datei `admin-language.component.html`

```
1  <fieldset class="intmanAdminLanguage">
2    <legend>Spracheinstellungen: {{lang.title}}</legend>
3    <p>Sprach-ID nach <a
     ↪  href="https://tools.ietf.org/html/rfc1766">RFC1766</a>:
     ↪  <strong>{{lang.id}}</strong></p>
4    <p><label>Bezeichnung: <input type="text"
     ↪  [(ngModel)]="lang.title" (keyup)="detectChange()"
     ↪  (input)="detectChange()"
     ↪  (change)="detectChange()"></label></p>
```

```
5    <p *ngIf="!isDefLang"><label><input type="checkbox"
     ↪  [(ngModel)]="lang.selectable"
     ↪  (change)="detectChange()">Sprache ist auswählbar</label></p>
6    <p *ngIf="isDefLang"><strong>Es handelt sich um die
     ↪  Standardsprache.</strong></p>
7    <p *ngIf="!isDefLang && lang.selectable"><label>Meldung bei
     ↪  nicht verfügbarer Übersetzung: <textarea
     ↪  [(ngModel)]="lang.unavailableText" (keyup)="detectChange()"
     ↪  (input)="detectChange()"
     ↪  (change)="detectChange()"></textarea></label></p>
8    <p class="intman-language-buttons intman-buttons"><button
     ↪  *ngIf="changedSomething" (click)="save()">Änderungen
     ↪  speichern!</button><button *ngIf="changedSomething"
     ↪  (click)="reset()">Änderungen zurücksetzen</button></p>
9    <p *ngIf="!isDefLang" class="intman-delete-buttons
     ↪  intman-buttons">
10     <button *ngIf="!deletionRequested"
       ↪  (click)="requestDeletion()">Sprache löschen</button>
11     <span *ngIf="deletionRequested && !deletionPossible">Sie
       ↪  müssen zunächst alle Übersetzungen in dieser Sprache
       ↪  löschen!</span>
12     <span *ngIf="deletionRequested && deletionPossible">Sind Sie
       ↪  sicher, dass Sie die Sprache
       ↪  <strong>{{lang.title}}</strong> löschen möchten?</span>
13     <button *ngIf="deletionRequested && deletionPossible"
       ↪  (click)="deletionRequested=false">Nein, nicht
       ↪  löschen!</button>
14     <button *ngIf="deletionRequested && deletionPossible"
       ↪  (click)="deleteLanguage()">Ja, {{lang.title}}
       ↪  löschen!</button>
15   </p>
16 </fieldset>
```

Das Template der `AdminLanguageComponent` ist geprägt durch Formularelemente, die je nach Zustand (*deletionRequested* oder *deletionPossible*) und *changedSo-*

mething) sichtbar sind oder auch nicht. Die Formularelemente ihrerseits verändern bei Betätigung oder Veränderung den Zustand. Sollte es sich um die unveränderbare Standardsprache handeln, wird statt der Formularelemente ein Hinweis eingeblendet.

A.2.6 Interne Komponente AdminTranslationComponent

Listing 2.12: Quelltext der Datei `admin-translation.component.ts`

```
1  import { Component, OnInit, Input, OnDestroy, HostListener,
   ↪  OnChanges } from '@angular/core';
2  import { TextContainer } from '../text-container';
3  import { Language } from '../language';
4  import { IntManLibService } from '../int-man-lib.service';
5  import { ContainerSetting } from '../container-setting';
6  import { Translation } from '../translation';
7
8  @Component({
9    selector: 'intman-admin-translation',
10   templateUrl: './admin-translation.component.html',
11   styleUrls: ['./admin-translation.component.css']
12 })
13 export class AdminTranslationComponent implements OnInit,
   ↪  OnDestroy, OnChanges {
14
15   @Input() lang: Language;
16   @Input() containerSetting: ContainerSetting;
17
18   public isDefLang = false;
19   public notMatching = false;
20   public altLangDisplayed: Language;
21   public changedSomething = false;
22   public deletionRequested = false;
```

```
23    public translationContents: string[] = Array();
24    public altLangPreferred: boolean[] = Array();
25    public defTexts: string[] = Array();
26
27    constructor(private intManLibService: IntManLibService) { }
28
29    ngOnInit() {
30      this.intManLibService.registerTranslationComponent(this);
31    }
32    ngOnDestroy() {
33      this.intManLibService.unregisterTranslationComponent(this);
34    }
35    ngOnChanges() {
36      // initialize vars
37      this.isDefLang = false; this.notMatching = false;
        ↪  this.altLangDisplayed = undefined; this.changedSomething =
        ↪  false;
38      this.deletionRequested = false;
39
40      // initialize content array with empty strings - defTexts
        ↪  don't need to be initialized
41      this.translationContents = Array(); this.altLangPreferred =
        ↪  Array(); this.defTexts = Array();
42      while (this.translationContents.length <
        ↪  this.containerSetting.contains) {
        ↪  this.translationContents.push(''); }
43      while (this.altLangPreferred.length <
        ↪  this.containerSetting.contains) {
        ↪  this.altLangPreferred.push(false); }
44
45      if (this.containerSetting.contains > 0) {
46        // gather translation
47        this.intManLibService.getTranslation(
          ↪  this.containerSetting.id,
          ↪  this.lang.id).subscribe(
```

```
48          t => {
49            if (t.length > 0) {
50              // always use right translation if available
51              const filtered = t.filter(trans => trans.langId ===
                ↪  this.lang.id);
52              if (filtered.length > 0) { t[0] = filtered[0]; }
53
54              // check whether translation language matches language
55              if (t[0].langId === this.lang.id) {
56                // check whether translation length matches
                  ↪  containerSetting
57                this.notMatching = t[0].contents.length !==
                  ↪  this.containerSetting.contains;
58                let contents = t[0].contents;
59                const altLangPreferred = t[0].preferAltLang.length
                  ↪  === this.containerSetting.contains ?
                  ↪  t[0].preferAltLang : Array();
60                if (this.notMatching) {
61                  if (contents.length >
                    ↪  this.containerSetting.contains) {
62                    // join all contents into first content
63                    contents = [contents.join('')];
64                  }
65                  // add empty contents to match setting
66                  while (contents.length <
                    ↪  this.containerSetting.contains) {
                    ↪  contents.push(''); }
67                  while (altLangPreferred.length <
                    ↪  this.containerSetting.contains) {
                    ↪  altLangPreferred.push(false); }
68                }
69
70                this.translationContents = contents;
71                this.altLangPreferred = altLangPreferred;
72              } else {
```

```
73                // register alternatively displayed language
74                this.intManLibService.getLanguage(t[
                   ↪  0].langId).subscribe(langDisplayed =>
                   ↪  this.altLangDisplayed = langDisplayed);
75              }
76            } else {
77              // register default language as alternatively
                 ↪  displayed language
78              this.intManLibService.defLang.subscribe(langDisplayed
                 ↪  => this.altLangDisplayed = langDisplayed);
79            }
80          }
81        );
82
83        // gather default translation
84        this.intManLibService.defLang.subscribe(
85          defLang => {
86            this.isDefLang = defLang.id === this.lang.id;
87            this.intManLibService.getTranslation(
               ↪  this.containerSetting.id,
               ↪  defLang.id).subscribe(
88              dT => {
89                if (dT.length > 0 && dT[0].contents.length ===
                   ↪  this.containerSetting.contains) {
90                  this.defTexts = dT[0].contents;
91                }
92              }
93            );
94          }
95        );
96      }
97    }
98    /* returns the DOM path to some textualContent with given index
       ↪  */
99    public getDomPath(contentIndex: number): string {
```

```
100     let domPath = '';
101     let searchIndex = -1;
102
103     // get index of contentIndex-nth occurance of #text, return ''
        ↪  if there is none.
104     for (let countSearches = 0; countSearches <= contentIndex;
        ↪  countSearches++) {
105       searchIndex++;
106       searchIndex =
          ↪  this.containerSetting.domSignature.indexOf('#text',
          ↪  searchIndex);
107       if (searchIndex === -1) { return ''; }
108     }
109
110     // set substring before occurence as domPath
111     // e.g. <p>#text<strong>#text</strong><em>#text</em></p> -->
        ↪  <p>#text<strong>#text</strong><em>
112     domPath = this.containerSetting.domSignature.substr(0,
        ↪  searchIndex);
113
114     // delete all #text occurances from domPath
115     // e.g. <p>#text<strong>#text</strong><em> -->
        ↪  <p><strong></strong><em>
116     while (domPath.indexOf('#text') > -1) { domPath =
        ↪  domPath.replace('#text', ''); }
117
118     // delete all > occurances from domPath
119     // e.g. <p><strong></strong><em> --> <p<strong</strong<em
120     while (domPath.indexOf('>') > -1) { domPath =
        ↪  domPath.replace('>', ''); }
121
122     // split domPath into Array
123     // e.g. <p<strong</strong<em -->
        ↪  ['','p','strong','/strong','em']
124     const split = domPath.split('<');
```

```
125       domPath = '';
126
127       let endTags: string[] = Array();
128
129       while (split.length > 0) {
130         const current = split.pop();
131         // avoid empty strings
132         if (current === '') { continue; }
133         if (current.substr(0, 1) === '/') {
134           // if it's an end tag, register!
135           endTags.push(current.substr(1));
136         } else {
137           // if it's not an end tag, check whether there is an end
              ↪  to it
138           if (endTags.indexOf(current) > -1) {
139             // if there's an end, purge the end from endTags and do
                ↪  nothing
140             endTags = endTags.filter(endTag => endTag !== current);
141           } else {
142             // if there's no end, add tag to domPath
143             if (domPath === '') {
144               domPath = current;
145             } else {
146               domPath = current + ' => ' + domPath;
147             }
148           }
149         }
150       }
151
152       return domPath;
153     }
154     /** request deletion */
155     public requestDeletion() {
156       this.deletionRequested = true;
157     }
```

```
158   /** delete language on user input */
159   public deleteTranslation() {
160     const container = this.containerSetting;
161     if (this.deletionRequested) {
162       this.intManLibService.deleteTranslation(container.id + '-' +
        ↪  this.lang.id).subscribe(_ => {
163         // check whether we still need the container
164         this.intManLibService.checkDeleteContainer(container.id);
165       });
166     }
167   }
168   /** change status if there was some change */
169   public detectChange(): void { this.changedSomething = true; }
170   /** resets form data */
171   public reset(): void { this.ngOnChanges(); }
172   /* saves current form data to server */
173   public save(): void {
174     // assemble new Translation object
175     const newT = new Translation();
176     newT.id = this.containerSetting.id + '-' + this.lang.id;
177     newT.langId = this.lang.id;
178     newT.containerId = this.containerSetting.id;
179     newT.contents = this.translationContents;
180     newT.preferAltLang = this.altLangPreferred;
181     // save object to server
182     this.intManLibService.updateTranslation(newT).subscribe();
183   }
184 }
```

Die Komponente `AdminTranslationComponent` nimmt von der übergeordneten Komponente Sprach- und Textblockdaten in Form von `lang` und `containerSetting` auf, ermittelt (auch nach jeder Änderung) die zugehörigen Übersetzungsdaten durch den zentralen Service und stellt diese in einem Formular zur Bearbeitung dar. Die Komponente aktualisiert sich bei einer Veränderung selbsttätig und ist im zentralen Service registriert. Ein Speicher-Button wird angezeigt

sobald eine Veränderung detektiert wurde (`changedSomething`), das Löschen der Übersetzungsdaten ist erst nach nochmaliger Bestätigung (`deletionRequested` / `requestDeletion`) möglich. Bemerkenswert ist die Methode `getDomPath`, die in der Lage ist, zu jeder gegebenen Indexnummer aus der bekannten *domSignature* des Containers zu berechnen, welche Elemente an der Textstelle mit dieser Indexnummer wie ineinander verschachtelt sind.

Listing 2.13: Quelltext der Datei `admin-translation.component.html`

```
<fieldset class="intmanAdminTranslation">
  <legend>{{containerSetting.id}} in {{lang.title}}</legend>
  <p><strong>DOM-Signatur des Containers:</strong>
   ↪  <code>{{containerSetting.domSignature}}</code></p>
  <p *ngIf="isDefLang"><strong>Der Inhalt in der Standardsprache
   ↪  muss im Code der Anwendung angepasst werden.</strong></p>
  <p *ngIf="notMatching"><strong>Die Übersetzung passt nicht zur
   ↪  DOM-Signatur und muss korrigiert werden. Bitte passen Sie
   ↪  vor dem Speichern die Inhalte der Felder an:</strong></p>
  <p *ngIf="altLangDisplayed !== undefined"><strong>Es existiert
   ↪  aktuell keine eigene Übersetzung. Der Container wird deshalb
   ↪  nach <em>{{altLangDisplayed.title}}</em> übersetzt.
   ↪  </strong>Füllen Sie die Felder aus, um eine eigene
   ↪  Übersetzung zu erzeugen:</p>
  <p *ngFor="let defText of defTexts; let i = index">
    <label><strong><span
     ↪  [innerHtml]='getDomPath(i)'></span>:</strong>
     ↪  {{defText}}<textarea *ngIf="!isDefLang &&
     ↪  !altLangPreferred[i]" type="text"
     ↪  [(ngModel)]="translationContents[i]"
     ↪  (keyup)="detectChange()" (input)="detectChange()"
     ↪  (change)="detectChange()"></textarea></label>
```

```
9       <label *ngIf="!isDefLang"
        ↪  class="intman-alt-lang-option"><input type="checkbox"
        ↪  [(ngModel)]="altLangPreferred[i]"
        ↪  (change)="detectChange()"> Übersetzung in alternative
        ↪  Sprache bevorzugen</label>
10    </p>
11    <p class="intman-translation-buttons intman-buttons"><button
      ↪  *ngIf="changedSomething" (click)="save()">Änderungen
      ↪  speichern!</button><button *ngIf="changedSomething"
      ↪  (click)="reset()">Änderungen zurücksetzen</button></p>
12    <p *ngIf="defTexts.length > 0 && altLangDisplayed === undefined
      ↪  && !isDefLang" class="intman-delete-buttons intman-buttons">
13      <button *ngIf="!deletionRequested"
        ↪  (click)="requestDeletion()">Übersetzung löschen</button>
14      <span *ngIf="deletionRequested">Sind Sie sicher, dass Sie
        ↪  diese Übersetzung löschen möchten?</span>
15      <button *ngIf="deletionRequested"
        ↪  (click)="deletionRequested=false">Nein, nicht
        ↪  löschen!</button>
16      <button *ngIf="deletionRequested"
        ↪  (click)="deleteTranslation()">Ja, Übersetzung
        ↪  löschen!</button>
17    </p>
18  </fieldset>
```

Das Template der `AdminTranslationComponent` ist geprägt durch Formularelemente, die je nach Zustand (*deletionRequested* und *changedSomething*) sichtbar sind oder auch nicht. Die Formularelemente ihrerseits verändern bei Betätigung oder Veränderung den Zustand. Interessant ist an dieser Stelle vor allem die Darstellung der einzelnen Übersetzungsdaten: Zu jeder Textstelle innerhalb des Containers wird die standardsprachliche Übersetzung zusammen mit dem Pfad zur momentanen Verschachtelungsebene einer *textarea* gegenübergestellt. Sollte es sich um die Standardsprache handeln, deren Übersetzungen nur über die Inhalte des Quellcode

angepasst werden können, wird statt der Formularelemente ein Hinweis eingeblendet.

A.2.7 Die Direktive idDirective

Listing 2.14: Quelltext der Datei `id.directive.ts`

```
1  import { Directive, ElementRef, Input, OnInit, Renderer2,
   ↪  OnDestroy } from '@angular/core';
2  import { TextContainer } from './text-container';
3  import { IntManLibService } from './int-man-lib.service';
4
5  @Directive({
6    selector: '[intmanId]'
7  })
8  export class IdDirective implements OnInit, OnDestroy {
9    constructor(private el: ElementRef, private renderer: Renderer2,
     ↪  private intManLibService: IntManLibService) {  }
10
11   textContainer: TextContainer;
12   @Input() intmanId: string;
13
14   ngOnInit() {
15     this.textContainer = new TextContainer(this.intmanId,
       ↪  this.el.nativeElement, this.renderer,
       ↪  this.intManLibService);
16     this.intManLibService.getCurrentLanguage().subscribe(lang =>
       ↪  this.textContainer.switchLanguage(lang));
17   }
18   ngOnDestroy() { this.textContainer.destroy(); }
19 }
```

Bei der Direktive `idDirective` handelt es sich um eine Attribut-Direktive, die an jedem Element mit dem Attribut `intmanId` erzeugt wird. Sobald eine Instanz dieser

Direktive erzeugt wird, ruft sie ihrerseits einen *TextContainer* ins Leben, dem sie ihre `intmanId` sowie das native Element übergibt, dem sie zugeordnet ist. Am Ende ihres Lifecycle löst sie den `textContainer` mit auf.

A.2.8 Die Klasse TextContainer

Listing 2.15: Quelltext der Datei `text-container.ts`

```
1  import { ContainerSetting } from './container-setting';
2  import { IntManLibService } from './int-man-lib.service';
3  import { TextualContent } from './textual-content';
4  import { Renderer2 } from '@angular/core';
5  import { Language } from './language';
6  import { Observable } from 'rxjs';
7  import { map, tap } from 'rxjs/operators';
8  import { Translation } from './translation';
9
10 export class TextContainer {
11   public id: string;
12   public nativeElement: any;
13   public renderer: Renderer2;
14   public domSignature: string;
15   public currentLang: Language;
16   public altLangDisplayed: Language;
17   public contents: TextualContent[];
18   private containerSetting: ContainerSetting;
19   private altLangNotification: any;
20   private translationCache = { };
21   private defaultTranslations: string[];
22
23   constructor(id: string, nativeElement: any, renderer: Renderer2,
     ↪  private intManLibService: IntManLibService) {
24     this.id = id;
25     this.nativeElement = nativeElement;
```

```
26      this.renderer = renderer;
27      this.intManLibService.getCurrentLanguage().subscribe(
28        lang => {
29          this.currentLang = lang;
30          this.contents.forEach(cont =>
            ↪  cont.initiateLanguage(lang));
31        }
32      );
33      this.contents = [];
34      // calculate domSignature and gather textual contents
35      this.domSignature = this.exploreDOM(this.nativeElement);
36
37      this.intManLibService.registerContainer(this);
38
39      // load containerSetting or create it and update if necessary
40      this.intManLibService.getContainerSettings(
    this.id).subscribe(
41        setting => {
42          if (setting === undefined || this.domSignature ===
            ↪  setting.domSignature) {
43            const newSetting = new ContainerSetting();
44            newSetting.id = this.id;
45            newSetting.domSignature = this.domSignature;
46            newSetting.contains = this.contents.length;
47            if (setting === undefined) {
48              // create new setting
49              this.intManLibService.addContainerSetting(
                ↪  newSetting).subscribe(s =>
                ↪  {
50                this.containerSetting = s;
51                // trigger registration of default translation
52                this.registerDefaultTranslation(undefined,
                  ↪  (this.containerSetting.contains - 1));
53              });
54            } else {
```

```
55              // update setting
56              this.intManLibService.updateContainerSetting(this.id,
                ↪  newSetting).subscribe();
57              this.containerSetting = newSetting;
58              // trigger registration of default translation
59              this.registerDefaultTranslation(undefined,
                ↪  (this.containerSetting.contains - 1));
60            }
61          } else {
62            this.containerSetting = setting;
63            // trigger registration of default translation
64            this.registerDefaultTranslation(undefined,
                ↪  (this.containerSetting.contains - 1));
65
66          }
67        }
68      );
69    }
70
71    /* receive default translation from contents and interact with
      ↪  server when all are received */
72    public registerDefaultTranslation(text: string, index: number):
      ↪  void {
73      if (this.defaultTranslations === undefined) {
74        this.defaultTranslations = Array();
75      }
76      while (this.defaultTranslations.length <= index) {
        ↪  this.defaultTranslations.push(undefined); }
77      if (text !== undefined) { this.defaultTranslations[index] =
        ↪  text; }
78      // interact with server only after containerSetting is loaded
79      if (this.containerSetting !== undefined &&
        ↪  this.defaultTranslations.length ===
        ↪  this.containerSetting.contains) {
80        let registeredAll = true;
```

```
81        this.defaultTranslations.forEach(t => registeredAll =
          ↪  registeredAll && (t !== undefined));
82
83        if (registeredAll) {
84          // check whether service knows default translation, create
            ↪  otherwise, update on detected changes
85          this.intManLibService.defLang.subscribe(defLang => {
86            this.intManLibService.getTranslation(this.id,
              ↪  defLang.id).subscribe(defTranslation => {
87              // only use translations matching language exactly
88              defTranslation = defTranslation.filter(t => t.langId
                ↪  === defLang.id);
89
90              if (defTranslation.length === 0) {
91                // create default translation
92                const newTranslation = new Translation();
93                newTranslation.containerId = this.id;
94                newTranslation.langId = defLang.id;
95                newTranslation.id = this.id + '-' + defLang.id;
96                newTranslation.preferAltLang = Array();
97                while (newTranslation.preferAltLang.length <
                  ↪  this.containerSetting.contains) {
                  ↪  newTranslation.preferAltLang.push(false); }
98                newTranslation.contents = this.defaultTranslations;
99                this.intManLibService.addTranslation(
                  ↪  newTranslation).subscribe();
100             } else {
101               let identicalTexts = true;
102               // compare default translation to registered
                  ↪  translations
103               defTranslation[0].contents.forEach((t, i) =>
                  ↪  identicalTexts = identicalTexts && t ===
                  ↪  this.defaultTranslations[i]);
104
105               if (!identicalTexts) {
```

```
106                 defTranslation[0].contents =
                     ↪  this.defaultTranslations;
107                 this.intManLibService.updateTranslation(
                     ↪  defTranslation[0]).subscribe();
108               }
109             }
110           });
111         });
112       }
113     }
114   }
115
116   /**
117    * get translation for textual content
118    * filter translations from server for matching domSignature,
119    * return an empty observable to allow for an alternative
    ↪  language to be chosen
120    */
121   public getTranslations(langId: string):
      ↪  Observable<Translation[]> {
122     if (this.translationCache[langId] === undefined) {
123       this.translationCache[langId] =
          ↪  this.intManLibService.getTranslation(this.id,
          ↪  langId).pipe(
124         map(translations => translations.filter(translation =>
            ↪  translation.contents.length === this.contents.length))
125       );
126     }
127     return this.translationCache[langId];
128   }
129
130   /** this function is called upon destruction of the connected
      ↪  component */
131   public destroy(): void {
132     this.intManLibService.unregisterContainer(this);
```

```
133       while (this.contents.length > 0) {
134         this.contents.pop().destroy();
135       }
136       this.translationCache = {};
137     }
138
139     /** removes all translations from cache and triggers cache flush
        ↪  on all contents */
140     public flushTranslations(): void {
141       this.translationCache = { };
142       this.contents.forEach(cont => cont.flushTranslations());
143     }
144
145     /* Display the texts of all contents in another language. */
146     public switchLanguage (targetLanguage: Language): void {
147       // resets displayed language
148       if (this.altLangDisplayed !== undefined) {
149         this.altLangDisplayed = undefined;
150         this.renderer.removeChild(
          ↪  this.altLangNotification.parentNode,
          ↪  this.altLangNotification);
151         this.altLangNotification = undefined;
152       }
153       // set language on all contents
154       this.contents.forEach(element => {
155        element.switchLanguage(targetLanguage.id);
156       });
157       this.currentLang = targetLanguage;
158     }
159
160     /** Check all contents for whether there is still some content
        ↪  in the containers language and switch internally otherwise.
        ↪  */
161     public checkLanguage (): void {
162       let someContentStillSameLanguage = false;
```

```
163       this.contents.forEach(element => {
164         // compare to current Language or to displayed language
165         if ((this.currentLang !== undefined && element.currentLangId
            ↪  === this.currentLang.id) || (this.altLangDisplayed !==
            ↪  undefined && element.currentLangId ===
            ↪  this.altLangDisplayed.id)) {
166           someContentStillSameLanguage = true;
167         }
168       });
169
170       // Switch language to first contents language if there are no
          ↪  content parts left in original language
171       if (!someContentStillSameLanguage && this.contents.length > 0)
          ↪  {
172         this.intManLibService.getLanguage(this.contents[
            ↪  0].currentLangId).subscribe(lang => this.currentLang =
            ↪  lang);
173       }
174     }
175
176     /** In case the current language cannot be displayed because of
          ↪  missing translations and there is also no compatible
          ↪  translation, switch the whole container to an alternatively
          ↪  chosen language */
177     public switchToAlternativeLanguage(): void {
178       // only run if alternative language isn't already chosen
179       if (this.altLangDisplayed === undefined) {
180         // future implementations may use also other criteria for
            ↪  choosing alternative language
181         // for now, only use default language
182         // for default language, no translations have to be fetched
            ↪  from server as default translations are provided via
            ↪  source code
183         // for other languages the domSignature might have to be
            ↪  checked!
```

```
184       this.intManLibService.defLang.subscribe(defLang => {
185         // sets displayed language
186         this.altLangDisplayed = defLang;
187         // set language on all contents
188         this.contents.forEach(element => {
189           element.switchLanguage(defLang.id);
190         });
191
192         // show notification for alternative translation if not
            ↪ already showing
193         if (this.altLangNotification === undefined) {
194           this.altLangNotification =
              ↪ this.renderer.createElement('div');
195           this.renderer.appendChild(this.altLangNotification,
              ↪ this.renderer.createText('(' +
              ↪ this.currentLang.unavailableText + ')'));
196           this.renderer.addClass(this.altLangNotification,
              ↪ 'intmanAltNotification');
197           this.renderer.appendChild(this.nativeElement,
              ↪ this.altLangNotification);
198           this.renderer.setStyle(this.altLangNotification,
              ↪ 'color', 'red');
199         }
200
201       });
202     }
203   }
204
205   /** Log a message within the message service provided by the
      ↪ libService. */
206   public log(message: string): void {
207     this.intManLibService.log(message);
208   }
209
```

```
210   /** Internal Method to gather all textual contents as well as
      ↪  the DOM-Signature of the container. */
211   private exploreDOM(node: any): string {
212     let signature = '';
213     if (node.nodeName === '#text') {
214       // if it's a text node, generate new textual content object
        ↪  and register it
215       const textualContent = new TextualContent(this,
        ↪  this.contents.length, node, this.intManLibService);
216       this.contents.push(textualContent);
217       signature += node.nodeName;
218     } else {
219       if (node.nodeName.substr(0, 1) !== '#') {
220         signature += '<' + node.nodeName + '>';
221       }
222     }
223     // if there are child nodes, invoke method recursively
224     node.childNodes.forEach(cur => signature +=
      ↪  this.exploreDOM(cur));
225     if (node.nodeName.substr(0, 1) !== '#') {
226       signature += '</' + node.nodeName + '>';
227     }
228     return signature;
229   }
230
231 }
```

Die Instanz der Klasse `TextContainer` repräsentiert eine zusammenhängende, zu übersetzende Passage mit einer einzigen *intmanId*, die unter `id` abgelegt wird. Nach der Initialisierung wird mit der Methode `exploreDom` die Verschachtelung des `nativeElement` durchlaufen; dabei werden an geeigneten Stellen neue Instanzen von *TextualContent* unter `contents` angelegt. Diese melden ihrerseits die standardsprachlichen Texte an den *TextContainer* zurück, der diese in der Methode `registerDefaultTranslations` entgegennimmt, sammelt und

schließlich nach Überprüfung auf Änderungen gesammelt dem zentralen Service zur Verfügung stellt. Die Klasse gibt ihr bekannt gewordene Sprachwechsel an die *contents* weiter, fordert Übersetzungen am zentralen Service an und speichert diese zwischen. Sollte aufgrund fehlender Übersetzungen ein Wechsel in die Standardsprache notwendig sein, koordiniert die Klasse diesen Sprachwechsel mittels `switchToAlternativeLanguage` und zeigt am nativen Element einen Hinweistext an.

A.2.9 Die Klasse TextualContent

Listing 2.16: Quelltext der Datei `textual-content.ts`

```
1  import { IntManLibService } from './int-man-lib.service';
2  import { Translation } from './translation';
3  import { TextContainer } from './text-container';
4  import { Language } from './language';
5
6  export class TextualContent {
7    container: TextContainer;
8    id: string;
9    index: number;
10   currentTextNode: any;
11   translatedTextNodes: {};
12   currentLangId: string;
13
14   constructor(container: any, index: number, currentTextNode: any,
     ↪  private intManLibService: IntManLibService) {
15     this.container = container;
16     this.index = index;
17     this.currentTextNode = currentTextNode;
18     this.translatedTextNodes = {};
19     // either use container language or do nothing as the
       ↪  container will inform the content whenever it gets defined
```

```
20      if (container.lang !== undefined) {
21        this.initiateLanguage(container.lang);
22      }
23      // generate id from container id
24      this.id = container.id + '-' + index;
25    }
26
27    /** encapsulates initiation of language as there is no way of
      ↪  being sure that language is already loaded on creation */
28    public initiateLanguage(lang: Language): void {
29      this.currentLangId = lang.id;
30      // Use default text as translation for default language
31      const currentTextNode = this.currentTextNode;
32      this.translatedTextNodes[this.currentLangId] =
        ↪  currentTextNode;
33      // register default translation at container
34      this.container.registerDefaultTranslation(
        ↪  currentTextNode.nodeValue,
        ↪  this.index);
35    }
36    /** this function is called upon destruction of the connected
      ↪  container */
37    public destroy(): void {
38      this.container = undefined;
39      this.translatedTextNodes = {};
40    }
41    /** Display the contents of the associated element in another
      ↪  language. */
42    public switchLanguage (targetLangId: string): void {
43      // never run if translation and native Element are same -
        ↪  otherwise text will disappear!
44      if (this.translatedTextNodes[targetLangId] !==
        ↪  this.currentTextNode) {
45        // use translation from storage or get from service
46        if (this.translatedTextNodes[targetLangId] !== undefined) {
```

```
47          // replace content with translated content
48          const renderer = this.container.renderer;
49          const parent = this.currentTextNode.parentNode;
50          renderer.insertBefore(parent,
            ↪  this.translatedTextNodes[targetLangId],
            ↪  this.currentTextNode);
51          renderer.removeChild(parent, this.currentTextNode);
52          this.currentTextNode =
            ↪  this.translatedTextNodes[targetLangId];
53          this.currentLangId = targetLangId;
54          // trigger check for containers language
55          this.container.checkLanguage();
56        } else {
57          // ask container for translation and run once again
58          this.container.getTranslations(targetLangId).subscribe(
            ↪  translations =>
            ↪  {
59            // always use right translation if available - e.g.
               ↪  avoids getting translation for en-GB when searching
               ↪  for en
60            const filtered = translations.filter(trans =>
               ↪  trans.langId === targetLangId);
61            if (filtered.length > 0) { translations[0] =
               ↪  filtered[0]; }
62            // respect setting for alternative language if
               ↪  applicable
63            const altLangPreferred = translations.length === 1 &&
64              translations[0].preferAltLang.length ===
                 ↪  translations[0].contents.length ?
                 ↪  translations[0].preferAltLang[this.index] : false;
65            if (translations.length === 1 && !altLangPreferred) {
66              this.translatedTextNodes[targetLangId] =
                 ↪  this.container.renderer.createText(translations[
                 ↪  0].contents[this.index]);
67              this.switchLanguage(targetLangId);
```

```
68              } else {
69                // fallback no. 1 - if there is no translation, try to
                  ↪ find another compatible language
70                // as languages starting with i- or x- aren't
                  ↪ necessarily related to one another (RFC1766), only
                  ↪ use 2-letter-codes
71                if (targetLangId.substr(0, 2) !== 'i-' &&
                  ↪ targetLangId.substr(0, 2) !== 'x-') {
72                  this.container.getTranslations(targetLangId.substr(
                    ↪ 0, 2)).subscribe(compTrans =>
                    ↪ {
73                    // filter out this language (in case an
                      ↪ alternative language is preferred)
74                    compTrans = compTrans.filter(trans => trans.langId
                      ↪ !== targetLangId);
75
76                    if (compTrans.length > 0) {
77                      // prefer universal phrases
78                      let chosenTrans: Translation;
79                      compTrans.forEach(t => { if (t.langId ===
                        ↪ targetLangId.substr(0, 2)) { chosenTrans =
                        ↪ t; }});
80                      if (chosenTrans === undefined) { chosenTrans =
                        ↪ compTrans[0]; }
81                      // switch this textual content to compatible
                        ↪ language
82                      // construct new Language to ensure no
                        ↪ alternative language of this language is
                        ↪ chosen
83                      this.switchLanguage(chosenTrans.langId);
84                    } else {
85                      // fallback no. 2 - if there is no compatible
                        ↪ language for a single textual content,
```

```
86                    // switch container to alternative language in
                      ↪  order to keep the containers context
                      ↪  together
87                    // if the alternative Language was preferred,
                      ↪  don't switch container but fall back to
                      ↪  default language for single content
88                    if (!altLangPreferred) {
89                      this.container.switchToAlternativeLanguage();
90                    } else {
91                      this.intManLibService.defLang.subscribe(
                        ↪  defLang =>
                        ↪  this.switchLanguage(defLang.id));
92                    }
93                  }
94                });
95              }
96            }
97          });
98        }
99      }
100   }
101   /** Log a message within the message service provided in its
      ↪  container. */
102   public log(message: string): void {
103     this.container.log(message);
104   }
105   /** reset translation cache and display new texts */
106   public flushTranslations(): void {
107     this.translatedTextNodes = { };
108     this.switchLanguage(this.currentLangId);
109   }
110 }
```

Die Instanz der Klasse `TextualContent` repräsentiert eine einzelne Textstelle innerhalb einer verschachtelten Struktur und

```
ist dieser Passage, ihrem container zugeordnet. Intern
hält sie verschiedene TextNodes mit den ihr bereits
bekannten Übersetzungen vor, die sie beim Sprachwechsel
mittels der Methode switchLanguagegegeneinander austauscht.
Sollte die Instanz keine Übersetzung für diese Sprache
kennen, so fordert sie selbige zunächst an oder weicht bei
nicht-Existenz auf eine Alternativübersetzung aus einer
anderen Sprachvariation aus.
```

A.2.10 Zentraler Service IntManLibService

Der `IntManLibService` stellt viele Helferfunktionalitäten bereit; unter anderem wickelt er den gesamten HTTP-Netzwerkverkehr ab. Das bedingt eine große Häufung ähnlich implementierter Funktionen, was den Quelltext des Service in die Länge zieht. Daher wird im Folgenden bei vielen Methoden des Service nur exemplarisch eine Implementierung gezeigt und bei den anderen Variationen für den Zweck des Abdrucks in dieser Arbeit die konkrete Implementierung ausgelassen; stattdessen wird nur die Signatur der Methode abgedruckt. Aus Gründen der Übersichtlichkeit wurden die Methoden außerdem teilweise anders angeordnet als in der realen Implementierung.

Listing 2.17: Quelltext der Datei `int-man-lib.service.ts`

```
1 import { TextContainer } from './text-container';
2 import { Injectable } from '@angular/core';
3 import { Observable, of } from 'rxjs';
4 import { catchError, tap, flatMap, map } from 'rxjs/operators';
5 import { HttpClient, HttpHeaders } from '@angular/common/http';
6 import { Language } from './language';
7 import { Translation } from './translation';
8 import { AdminTranslationComponent } from
  ↪ './admin-translation/admin-translation.component';
9 import { AdminLanguageComponent } from
  ↪ './admin-language/admin-language.component';
```

```
10 import { AdminComponent } from './admin/admin.component';
11 import { SwitcherComponent } from './switcher/switcher.component';
12 import { ContainerSetting } from './container-setting';
13
14 const httpOptions = {
15   headers: new HttpHeaders({ 'Content-Type': 'application/json'})
16 };
17
18 @Injectable({
19   providedIn: 'root'
20 })
21 export class IntManLibService {
22
23   private dataUrl = 'api';
24   public defLang: Observable<Language>;
25   private curLang: Language;
26   private containers: TextContainer[] = [];
27   private translationComponents: AdminTranslationComponent[] = [];
28   private languageComponents: AdminLanguageComponent[] = [];
29   private adminComponents: AdminComponent[] = [];
30   private switcherComponents: SwitcherComponent[] = [];
31
32   /** Register some text container to the general service in order
     ↪  to receive updates */
33   public registerContainer(which: TextContainer): void {
     ↪  this.containers.push(which); }
34
35   /** Unregister some text container from the general service in
     ↪  order to receive updates */
36   public unregisterContainer(which: TextContainer): void {
     ↪  this.containers = this.containers.filter(comp => comp !==
     ↪  which); }
37
38   public registerSwitcherComponent(which: SwitcherComponent): void
     ↪  { /* ... */ }
```

```
39          public unregisterSwitcherComponents(which:
            ↪  SwitcherComponent): void { /* ... */ }
40      public registerTranslationComponent(which:
        ↪  AdminTranslationComponent): void { /* ... */ }
41      public unregisterTranslationComponent(which:
        ↪  AdminTranslationComponent): void { /* ... */ }
42      public registerLanguageComponent(which: AdminLanguageComponent):
        ↪  void { /* ... */ }
43      public unregisterLanguageComponent(which:
        ↪  AdminLanguageComponent): void { /* ... */ }
44      public registerAdminComponent(which: AdminComponent): void { /*
        ↪  ... */ }
45      public unregisterAdminComponent(which: AdminComponent): void {/*
        ↪  ... */ }
46
47      /** gets the current language the application shall be served in
        ↪  */
48      public getCurrentLanguage(): Observable<Language> {
49        if (this.curLang === undefined) {
50          return this.defLang;
51        } else {
52          return of(this.curLang);
53        }
54      }
55      /** sets the current language of the application to the given
        ↪  one */
56      public setLanguage(to: Language): void {
57        this.curLang = to;
58        this.containers.forEach(container => {
59          container.switchLanguage(to);
60        });
61        this.switcherComponents.forEach(comp => {
62          comp.lang = to.id;
63        });
64      }
```

```
65    /** gets a single language from server by server id */
66    public getLanguage(id: string): Observable<Language> {
67      const url = `${this.dataUrl}/languages/${id}`;
68      return this.http.get<Language>(url)
69      .pipe(
70        tap(_ => this.log(`fetched language id=${id}`)),
71        catchError(this.handleError<Language>(`getLanguage
          ↪ id=${id}`))
72      );
73    }
74    /** gets all available languages from server */
75    public getLanguages(): Observable<Language[]> {
76      const url = `${this.dataUrl}/languages`;
77      return this.http.get<Language[]>(url)
78        .pipe(
79          tap(ls => this.log('fetched ' + ls.length + '
            ↪ languages')),
80          catchError(this.handleError('getLanguages', []))
81        );
82    }
83    /** gets all translations for a given container in a given
      ↪ language */
84    public getTranslation(containerId: string, langId: string):
      ↪ Observable<Translation[]> {
85      const url = `${this.dataUrl}/translations/?containerId=${
        ↪ containerId}&langId=${langId}`;
86      return this.http.get<Translation[]>(url)
87      .pipe(
88        tap(_ => this.log(`fetched translation for container
          ↪ id=${containerId} with language lang=${langId}`)),
89        catchError(this.handleError<Translation[]>(`getLanguage
          ↪ id=${containerId} with language lang=${langId}`))
90      );
91    }
92
```

```
93    /** gets all translations for a given container */
94    public getAllTranslations(containerId: string):
      ↪  Observable<Translation[]> { /* ... */ }
95    /** gets all translations for a given language */
96    public getTranslationsByLanguage(langId: string):
      ↪  Observable<Translation[]> { /* ... */ }
97    /** gets all container settings from server */
98    public getAllContainerSettings(): Observable<ContainerSetting[]>
      ↪  { /* ... */ }
99    /** gets settings for a specific container from server */
100   public getContainerSettings(containerId: string):
      ↪  Observable<ContainerSetting> { /* ... */ }
101
102   /** POSTs a new Translation to the server */
103   public addTranslation(newTranslation: Translation):
      ↪  Observable<Translation> {
104     const url = `${this.dataUrl}/translations`;
105     return this.http.post<Translation>(url, newTranslation,
        ↪  httpOptions).pipe(
106       tap((t: Translation) => this.log(`translation with id
          ↪  ${t.id} saved to server.`)),
107       // register new translation
108       tap((t: Translation) => this.translationComponents
109         .filter(tC => t.langId === tC.lang.id && t.containerId ===
          ↪  tC.containerSetting.id)
110         .forEach(tC => tC.ngOnChanges())),
111       // flush cached translations if lang is not default lang
112       tap((t: Translation) => { this.defLang.subscribe(defLang =>
          ↪  { if (defLang.id !== t.langId) {
113         this.containers.forEach(c => c.flushTranslations());
114       } } ); } ),
115       // gather missing translations
116       tap(_ => this.adminComponents.forEach(aC =>
          ↪  aC.collectMissingTranslations())),
117       catchError(this.handleError<Translation>('addTranslation'))
```

```
118       );
119     }
120     /** PUTs an updated Translation to the server */
121     public updateTranslation(newTranslation: Translation):
        ↪  Observable<any> {
122       const url = `${this.dataUrl}/translations`;
123       return this.http.put<any>(url, newTranslation,
          ↪  httpOptions).pipe(
124         tap(_ => this.log(`translation with id ${newTranslation.id}
            ↪  updated on server.`)),
125         // register new translation
126         tap(_ => this.translationComponents
127           .filter(tC => newTranslation.langId === tC.lang.id &&
              ↪  newTranslation.containerId === tC.containerSetting.id)
128           .forEach(tC => tC.ngOnChanges())),
129         // flush cached translations if lang is not default lang
130         tap(_ => { this.defLang.subscribe(defLang => { if
            ↪  (defLang.id !== newTranslation.langId) {
131           this.containers.forEach(c => c.flushTranslations());
132         } } ); } ),
133         // gather missing translations
134         tap(_ => this.adminComponents.forEach(aC =>
            ↪  aC.collectMissingTranslations())),
135         catchError(this.handleError<any>('updateTranslation'))
136       );
137     }
138     /** DELETEs a translation from the server */
139     public deleteTranslation (translation: Translation | string):
        ↪  Observable<Translation> {
140       const id = typeof translation === 'string' ? translation :
          ↪  translation.id;
141       const url = `${this.dataUrl}/translations/${id}`;
142
143       return this.http.delete<Translation>(url, httpOptions).pipe(
```

```
144         tap(_ => this.log(`Translation with ${id} deleted from
            ↪ server.`)),
145         // reload translation Components
146         tap(_ => this.translationComponents.filter(item =>
            ↪ item.containerSetting.id + '-' + item.lang.id === id)
147                                             .forEach(item => {
                                                ↪ item.ngOnChanges();
                                                ↪ })),
148         // reset view of all language components
149         tap(_ => this.languageComponents.forEach(item =>
            ↪ item.deletionRequested = false)),
150         // flush all translations to request up to date translations
151         tap(_ => this.containers.forEach(item => {
            ↪ item.flushTranslations(); })),
152         // gather missing translations
153         tap(_ => this.adminComponents.forEach(aC =>
            ↪ aC.collectMissingTranslations())),
154         catchError(this.handleError<Translation>(␣
            ↪ 'deleteTranslation'))
155       );
156     }
157
158     /** POSTs a new Language to the server */
159     public addLanguage(newLang: Language): Observable<Language> { /*
        ↪ ... */ }
160     /** PUTs an updated Language to the server */
161     public updateLanguage(newLang: Language): Observable<any> { /*
        ↪ ... */ }
162     /** DELETEs a language from the server without checking
        ↪ implications. */
163     public deleteLanguage (lang: Language | string):
        ↪ Observable<Language> { /* ... */ }
164
165     /** POSTs a new ContainerSetting to the server */
```

```
166   public addContainerSetting(containerSetting: ContainerSetting):
      ↪  Observable<ContainerSetting> { /* ... */ }
167   /** PUTs an updated ContainerSetting to the server */
168   public updateContainerSetting(id: string, newSetting:
      ↪  ContainerSetting): Observable<any> { /* ... */ }
169   /** DELETEs a ContainerSetting from the server without checking
      ↪  implications */
170   public deleteContainerSetting (cS: ContainerSetting | string):
      ↪  Observable<ContainerSetting> { /* ... */ }
171
172   /** checks whether we still need a container setting, deletes it
      ↪  otherwise */
173   public checkDeleteContainer (containerId: string) {
174     this.defLang.subscribe(defLang => {
175       this.getAllTranslations(containerId).subscribe(translations
          ↪  => {
176         // delete container if all translations are in standard
            ↪  language and container is not currently registered
177         if (translations.filter(trans => trans.langId !==
            ↪  defLang.id).length === 0) {
178           let containerRegistered = false;
179           this.containers.forEach(c => { if (c.id === containerId)
              ↪  { containerRegistered = true; } });
180           if (!containerRegistered) {
181             this.deleteContainerSetting(containerId).subscribe();
182           }
183         }
184       });
185     });
186   }
187
188   /** Handle Http operation that failed. Let the app continue. --
      ↪  Taken from AngularTourOfHeroes */
189   private handleError<T> (operation = 'operation', result?: T) {
190     return (error: any): Observable<T> => {
```

```
191         // TODO: send the error to remote logging infrastructure
192         // console.error(error); // log to console instead
193         // TODO: better job of transforming error for user
            ↪  consumption
194         this.log(`${operation} failed: ${error.message}`);
195         // Let the app keep running by returning an empty result.
196         return of(result as T);
197       };
198     }
199
200     /** Log a message. */
201     public log(message: string) {
202       // console.log('Message: ' + message);
203     }
204
205     /** Creates an instance of IntManLibService. */
206     constructor(private http: HttpClient) {
207       this.http = http;
208       // get default language from server
209       const url = `${this.dataUrl}/defLangId`;
210       this.defLang = this.http.get<string>(url)
211       .pipe(
212         tap(langId => this.log(`fetched default language for app:
            ↪  ${langId}`)),
213         catchError(this.handleError<string>(`IntManLibService
            ↪  constructor - default language`)),
214         flatMap(langId => this.getLanguage(langId))
215       );
216     }
217   }
```

Die verkürzte Darstellung des zentralen Service `IntManLibService` gibt einen Überblick über seine Funktionalitäten: `curLang` und `defLang` beinhalten die aktuell gewählte Sprache und die Standardsprache. Der Service verfügt außerdem über Verzeichnisse der tatsächlich momentan aktiven Elemente der Bibliothek. Diese

werden durch `register`- und `unregister`-Methoden bedient. Für die via REST-API bereitgestellten Daten der Typen *Language*, *ContainerSetting* und *Translation* führt der Service entsprechende HTTP-Requests durch, wobei bei den HTTP-Methoden *POST*, *PUT* und *DELETE* jeweils die registrierten Elemente über die Änderung informiert und zu einer Aktualisierung angeregt werden.

A.3 Die HttpClientInMemoryWebApi der Testumgebung

Listing 3.1: Quelltext der Datei `in-memory-data.service.ts`

```
1  import { Injectable } from '@angular/core';
2  import { InMemoryDbService } from 'angular-in-memory-web-api';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class InMemoryDataService implements InMemoryDbService {
8
9    createDb() {
10     const db = {
11       defLangId : 'de-DE',
12       languages : [
13         { id: 'de-DE', title: 'Deutsch (Deutschland)',
          ↪  unavailableText: 'Eine Übersetzung in Deutsch ist
          ↪  leider nicht vorhanden.', selectable: true },
14         { id: 'en-GB', title: 'English (Great Britain)',
          ↪  unavailableText: 'Unfortunately there\'s no
          ↪  translation in English.', selectable: true },
15         { id: 'en-US', title: 'English (United States)',
          ↪  unavailableText: 'Unfortunately there\'s no
          ↪  translation in English.', selectable: true  },
```

```
16          { id: 'fr-FR', title: 'Français (France)',
              ↪ unavailableText: 'Malheureusement, il n\'y a pas de
              ↪ traduction.', selectable: true },
17          { id: 'de', title: 'Deutsch', unavailableText: '',
              ↪ selectable: false },
18          { id: 'en', title: 'English', unavailableText: '',
              ↪ selectable: false },
19          { id: 'fr', title: 'Français', unavailableText: '',
              ↪ selectable: false }
20        ],
21        translations : [
22          { id: 'IntManInternalChooseLanguage-en', containerId:
              ↪ 'IntManInternalChooseLanguage', langId: 'en',
              ↪ contents: ['Please choose your language:'],
              ↪ preferAltLang: [false]},
23          { id: 'IntManInternalChooseLanguage-fr', containerId:
              ↪ 'IntManInternalChooseLanguage', langId: 'fr',
              ↪ contents: ['Choisissez votre langue:'], preferAltLang:
              ↪ [false]},
24          { id: 'CasaTitle-de-DE', containerId: 'CasaTitle', langId:
              ↪ 'de-DE', contents: ['Willkommen bei Casa Del Diavolo -
              ↪ Pizza-Lieferdienst!'], preferAltLang: [false] },
25          { id: 'CasaTitle-en-GB', containerId: 'CasaTitle', langId:
              ↪ 'en-GB', contents: ['Welcome to Pizza-Service Casa Del
              ↪ Diavolo!'], preferAltLang: [false] },
26          { id: 'CasaSubtitle-en-GB', containerId: 'CasaSubtitle',
              ↪ langId: 'en-GB', contents: ['Owner: ', 'Mario
              ↪ Manichino'], preferAltLang: [false] }
27          // and many more translations for testing reasons...
28
29        ],
30        containerSettings : [
31          { id: 'CasaTitle', domSignature: '<H1>#text</H1>',
              ↪ contains: 1 },
```

```
32          { id: 'CasaSubtitle', domSignature: '<P>#text</P>',
            ↪  contains: 1 },
33          { id: 'NotUsed', domSignature: '<span>#text</span>',
            ↪  contains: 1 }
34        ]
35      };
36      return db;
37    }
38  }
```

Der `InMemoryDataService` ist Teil der Testumgebung *Casa del Diavolo* und liefert die Daten über simulierte HTTP-Requests an die Bibliothek. In `defLangId` wird ein String mit dem Tag der Standardsprache vorgehalten. Unter `languages` liegen Daten vom Typ *Language*, unter `translations` liegen Daten vom Typ *Translation* und unter `containerSettings` liegen Daten vom Typ *containerSetting*. Es handelt sich hier ganz bewusst um einen unvollständigen Datensatz, teilweise mit Inkonsistenzen, um innerhalb der Testumgebung alle Funktionalitäten der Bibliothek schnell testen zu können.

Listing 3.2: Quelltext der Datei `app.module.ts`

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { AppComponent } from './app.component';
4  import { IntManLibModule } from 'int-man-lib';
5  import { SwitcherComponent, IdDirective } from
   ↪  'projects/int-man-lib/src/public_api';
6  import { InMemoryDataService } from './in-memory-data.service';
7  import { HttpClientInMemoryWebApiModule } from
   ↪  'angular-in-memory-web-api';
8
9  @NgModule({
10   declarations: [
11     AppComponent
12   ],
```

```
13    imports: [
14      BrowserModule,
15      IntManLibModule,
16      // The HttpClientInMemoryWebApiModule module intercepts HTTP
        ↪  requests
17      // and returns simulated server responses.
18      // Remove it when a real server is ready to receive requests.
19      HttpClientInMemoryWebApiModule.forRoot(
20        InMemoryDataService, { dataEncapsulation: false }
21      )
22    ],
23    providers: [],
24    bootstrap: [AppComponent]
25  })
26  export class AppModule { }
```

Das *ngModule* der Anwendung zeigt die Einbindung des *InMemoryDataService* als simulierte HTTP-Schnittstelle durch die `HttpClientInMemoryWebApi`. Die Anwendung importiert `IntManLibModule` und sorgt dadurch dafür, dass der simulierte *HttpClient* via *dependency injection* in das Modul der Bibliothek injiziert wird.

A.4 Finale Version – Testumgebung Casa del Diavolo

Listing 4.1: Quelltext der Datei `casafinal.html`

```
1  <main style="text-align:center">
2    <header>
3      <h1 intmanId="CasaTitle">
4        Willkommen bei {{ title }}!
5      </h1>
6      <p intmanId="CasaSubtitle">{{subtitle}}</p>
7    </header>
```

```
8   <intman-switcher variant="dropdown"></intman-switcher>
9   <p intmanId="CasaDescription">Unsere Pizza ist eine ganz eigene
    ↪  Kreation, die [...] eine Antwort. <strong>Die Hauptsache
    ↪  ist</strong>, dass alle unsere Pizzen <em>mit viel Liebe zum
    ↪  Detail</em> über den Ladentisch [...] Einzigartigkeit
    ↪  nachzuahmen.</p>
10  <h1 intmanId="CasaOrder">Online-Bestellung</h1>
11  <p intmanId="CasaOffer">Wählen Sie aus unserem reichhaltigen
    ↪  Angebot.</p>
12  <form method="GET" class="order">
13    <div class="meals">
14      <fieldset class="meal" *ngFor="let meal of meals; let
        ↪  m=index"  intmanId="CasaMeal{{m}}">
15        <legend>{{meal.name}}</legend>
16        <p><img src="{{meal.icon}}"/><span>{{
          ↪  meal.description}}</span></p>
17        <input type="hidden" [name]="'names['+m+']'"
          ↪  value="{{meal.name}}"/>
18        <p *ngIf="meal.extras.length>0">Wählen Sie zusätzlich aus
          ↪  folgenden Extras:</p>
19        <ul *ngIf="meal.extras.length>0">
20          <li *ngFor="let extra of meal.extras">
21            <input type="checkbox" [name]="'extras['+m+']'"
              ↪  value="{{extra}}"/><span>{{extra}}</span>
22          </li>
23        </ul>
24        <p>Gewünschte Anzahl: <input type="number"
          ↪  [name]="'quantity['+m+']'" value="0"/></p>
25      </fieldset>
26    </div>
27    <fieldset class="buttons">
28      <button>Bestellung aufgeben →</button>
29    </fieldset>
30  </form>
```

```
31 </main>
32 <intman-admin></intman-admin>
```

Endgültige Version des HTML-Templates für Casa del Diavolo, inklusive bilbiothekss-pezifischer Änderungen. Der Vergleich mit Listing 1.1 zeigt, dass der Code nur an sehr wenigen Stellen verändert wurde: Einmal wurde der `intman-switcher` mit `variant="dropdown"` eingefügt und am Ende des Codes (in einer Produktivumgebung wäre das im Adminbereich der Fall) die Komponente `intman-admin`. Zusätzlich wurde an insgesamt 6 Stellen eine `intmanId` eingefügt. Dieses Beispiel zeigt, wie wenig Veränderungen an der Anwendung vorgenommen werden müssen, um die Bibliothek ordnungsgemäß einzubinden.

Abbildungsverzeichnis

2.1 Casa del Diavolo - ohne Internationalisierung. Eigenes Werk. 16
2.2 Klassendiagramm zur Grundstruktur. Eigenes Werk. 24

3.1 Verschiedene Darstellungsvarianten von SwitcherComponent. Eigenes Werk. 32
3.2 Gesamtansicht der AdminComponent. Eigenes Werk. 36
3.3 Detailansicht der AdminLanguageComponent. Eigenes Werk. 37
3.4 Detailansicht der AdminTranslationComponent. Eigenes Werk. 39
3.5 Die Testumgebung in unterschiedlichen Sprachen. Eigenes Werk. 43

Quelltextverzeichnis

1.1 Quelltext der Datei `casabasis.html` 51
2.1 Quelltext der Datei `int-man-lib.module.ts` 53
2.2 Quelltext der Datei `public-api.ts` 54
2.3 Quelltext der Datei `container-setting.ts` 55
2.4 Quelltext der Datei `language.ts` 55
2.5 Quelltext der Datei `translation.ts` 56
2.6 Quelltext der Datei `switcher.component.ts` 56
2.7 Quelltext der Datei `switcher.component.html` 58
2.8 Quelltext der Datei `admin.component.ts` 59
2.9 Quelltext der Datei `admin.component.html` 63
2.10 Quelltext der Datei `admin-language.component.ts` 66
2.11 Quelltext der Datei `admin-language.component.html` 68
2.12 Quelltext der Datei `admin-translation.component.ts` 70
2.13 Quelltext der Datei `admin-translation.component.html` 77
2.14 Quelltext der Datei `id.directive.ts` 79

2.15 Quelltext der Datei `text-container.ts` 80
2.16 Quelltext der Datei `textual-content.ts` 89
2.17 Quelltext der Datei `int-man-lib.service.ts` 94
3.1 Quelltext der Datei `in-memory-data.service.ts` 103
3.2 Quelltext der Datei `app.module.ts` 105
4.1 Quelltext der Datei `casafinal.html` 106

Name: Janosch Walter Zoller Matrikelnummer: 753 134

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Janosch Walter Zoller